Wintersemester 2007/2008 und Sommersemester 2008

**Projektgruppe 517: Evakuierungsprobleme**

# Endbericht der Projektgruppe


Martin Groß Moukarram Kabbash
Jan-Philipp Kappmeier Sophia Kardung
Timon Kelter Joscha Kulbatzki Daniel Plümpe
Marcel Preuß Gordon Schlechter
Melanie Schmidt Sylvie Temme Matthias Woste


30. September 2008


Technische Universität Dortmund

Daniel Dressler & Prof. Dr. Martin Skutella
Fakultät II, Institut für Mathematik
Technische Universität Berlin

Markus Chimani & Karsten Klein
Lehrstuhl XI - Algorithm Engineering
Fakultät für Informatik
Technische Universität Dortmund


Evakuierungstool

# Inhaltsverzeichnis

# Vorwort

Evakuierungen von Gebäuden sind jedem von uns gut bekannt. Als Schüler erlebt man zahlreiche „Probealarme“, bei denen der optimale Fluchtweg und das Verhalten bei Feuer eingeübt wird. Dadurch soll erreicht werden, dass das Schulgebäude im Ernstfall schnell und mit wenig Panik geräumt werden kann.

Wie geht man aber mit Situationen um, in denen die betroffenen Personen zuvor nicht geschult werden können? Kann man den Einfluss von Panik und Verwirrung in einem solchen Fall im Vorhinein einschätzen? Und wie *plant* man ein Gebäude möglichst „evakuierungsfreundlich“? Vor dem Bau eines Gebäudes kann man keine Testevakuierungen durchführen, um den Einfluss von Gebäudestrukturen auf die durchschnittliche Evakuierungszeit abzuschätzen.

Um sich solchen Fragestellungen zu nähern, werden *Simulationen* verwendet. Das Verhalten von Personen in Stresssituationen wird dabei im Rechner abgebildet, so dass mit Hilfe virtueller Testläufe die Evakuierungszeit ermittelt werden kann. So kann man auch Gebäude in der Planung auf ihre Evakuierbarkeit testen. Wir verwenden zur Simulation *Zelluläre Automaten*. Diese basieren auf einfachen Grundannahmen, so dass das Modell überschaubar bleibt. Andererseits sind sie sehr mächtig und können gut erweitert werden.

Man kann Evakuierungsprobleme auch unter analytischen Gesichtspunkten betrachten und mit Hilfe von *Netzwerkflüssen* modellieren. Ein Netzwerkfluss stellt eine „optimale“ Evakuierung dar. Menschen werden hier durch einzelne Flusseinheiten dargestellt, deren Verhalten so aufeinander abgestimmt ist, dass alle Flusseinheiten das Gebäude in minimaler Zeit verlassen. Eine so berechnete Evakuierungszeit kann man als untere Schranke verwenden.

In der Projektgruppe 517 an der Technischen Universität Dortmund haben wir im Wintersemester 2007/2008 und im Sommersemester 2008 beide Ansätze zur

Modellierung von Evakuierungsproblemen verfolgt und auch eine Verbindung zwischen den Ansätzen hergestellt. Der vorliegende Bericht soll einen Einblick in unsere Arbeit geben und das von uns erstellte Tool *ZET* vorstellen. Dieses kann Evakuierungen simulieren und auf Basis von Netzwerkflüssen berechnen sowie beide Programmteile kombinieren.

# Teil I.

# Ablauf der Projektgruppe

# Inhaltsverzeichnis

# 1. Einleitung

## 1.1. Projektgruppen in Dortmund

Die Diplomprüfungsordnung Informatik in Dortmund sieht im Hauptstudium die Teilnahme an einer **Projektgruppe (PG)** vor. Eine PG besteht aus acht bis zwölf Studierenden, die sich unter Anleitung ein Jahr lang mit einem Thema beschäftigen. Das Ziel der Projektgruppe wird von den Veranstaltern im Projektgruppenantrag festgelegt.

Die Durchführung einer Projektgruppe beginnt mit einer **Seminarphase**. Jeder teilnehmende Student hält hier einen Vortrag über ein Teilgebiet des PG-Themas, damit alle einen Einstieg in die gemeinsame Arbeit und das Themengebiet finden.

Im Anschluss arbeiten die Studierenden selbstorganisiert daran, das Projektgruppenziel zu erreichen.

## 1.2. Teilnehmer der Projektgruppe

Unsere Projektgruppe wurde von

- Prof. Dr. Martin Skutella, Daniel Dressler
  Fakultät II - Mathematik und Naturwissenschaften, COGA
  Institut für Mathematik
  Technische Universität Berlin
- Markus Chimani, Karsten Klein
  Lehrstuhl XI - Algorithm Engineering
  Fakultät für Informatik
  Technische Universität Dortmund

betreut. Die Teilnehmer der Projektgruppe waren

- Martin Groß
- Moukarram Kabbash
- Jan-Philipp Kappmeier
- Sophia Kardung
- Timon Kelter
- Joscha Kulbatzki
- Daniel Plümpe
- Marcel Preuß
- Gordon Schlechter
- Melanie Schmidt
- Sylvie Katharina Temme
- Matthias Woste.

## 1.3. Minimalziel der Projektgruppe

Auszug aus dem Projektgruppenantrag:

> Es sollen mindestens zwei exakte Algorithmen im Bereich der Netzwerkflussprobleme im Hinblick auf Evakuierungsprobleme implementiert und getestet werden. Des weiteren soll zumindest eine einfache Computersimulation im Bereich der Personenevakuierung erstellt werden. Hierbei müssen einige spezielle Verhaltensmuster von Personen in Gefahrsituationen modelliert sein. Die Anpassung der simulierten Umwelt aufgrund einer Analyse der Ergebnisse der exakten Algorithmen soll zu einer Annäherung der Ergebnisse beider Methoden führen. Eine einfache graphische Oberfläche, die den Netzwerkgenerator integriert und sowohl Lösungen der Optimierungsverfahren als auch der Simulationsverfahren veranschaulicht, wird erwartet.

# 2. Seminarphase

Unser Projektgruppenthema teilt sich grundsätzlich in die beiden Bereiche Simulation und Optimierung auf. Dementsprechend bestand unser Auftaktseminar aus Vorträgen zu diesen beiden Themen, wobei Flüsse und Optimierung einen leicht größeren Anteil hatten.

Die Seminarphase fand von Montag, 15. Oktober 2007, bis Montag, 5. November 2007, statt. Pro Termin wurden zwei Vorträge gehalten, die Zeitplanung findet sich in Tabelle 2.1. Im Folgenden werden zunächst die Inhalte der Vorträge zusammengefasst, die sich mit Simulation beschäftigen. Im Anschluss finden sich Beschreibungen der Vorträge über Flussprobleme und -algorithmen. Innerhalb der beiden Abschnitte sind die Vorträge chronologisch geordnet.

## 2.1. Vorträge über Simulationsthemen

### 2.1.1. Einflüsse auf Evakuierungen & Evakuierungstools

**Parameter & Einflüsse auf Evakuierungen** Bei der Berechnung der Evakuierungszeit sollten verschiedene Dinge berücksichtigt werden. Zunächst sind die Art des zu evakuierenden Bereichs (z.B. Schiff, Wohnhaus, Einkaufszentrum) und dessen Eigenschaften wichtig. Notausgangsschilder, Feuermelder und Ähnliches sind für die Wegfindung zum Ausgang entscheidend.
Außerdem müssen die Eigenschaften der dort anzutreffenden Personen (z.B. Alter, Art der Kleidung oder Geduld sowie abgeleitete Eigenschaften wie Geschwindigkeit oder Alter) möglichst genau beschrieben werden. Mit dieser Problematik beschäftigt sich [30].

Das **Rimea-Protokoll** [1] ist eine Zusammenstellung von Richtlinien für Programme zur Berechnung von Evakuierungen. Dieses legt Vorschriften für die

| Termin | Vortragender | Thema |
|---|---|---|
| 15.10. | Marcel Preuß | Statische Flüsse:<br>Shortest Paths und Max Flow |
| | Sylvie Temme | Statische Flüsse:<br>Minimum Cost Flows |
| 19.10. | Joscha Kulbatzki | LP-Formulierungen<br>für dynamische Flussprobleme |
| | Sophia Kardung | Parameter bei Evakuierungen,<br>Überblick über existierende<br>Evakuierungssimulationstools |
| 22.10. | Timon Kelter | Lösungsansätze dynamischer Flüsse |
| | Jan-Philipp Kappmeier | Partikelsysteme |
| 26.10. | Daniel Plümpe | Lexmax Dynamic Flows und<br>Lexmax Earliest Arrival Flows |
| | Melanie Schmidt | Earliest Arrival Transshipment |
| 29.10. | Matthias Woste | Einführung zelluläre Automaten |
| | Gordon Schlechter | Einführung Agentensysteme |
| 05.11. | Martin Groß | Probleme mit zeit- und<br>flussabhängigen Fahrzeiten |
| | Moukarram Kabbash | Erweiterte zeitexpandierte Graphen |

**Tabelle 2.1.:** Zeitplan der Seminarphase

Genauigkeit der Simulation, die Art der zu betrachtenden Szenarien und die Wahl der Parameter (z.B. für die Altersverteilung) fest. Außerdem gibt es eine Anleitung zur Verifikation und Validierung von Simulationsprogrammen.

**Überblick über Simulationstools** In diesem Teil wurden einige Simulationstools vorgestellt. Diese Tools unterscheiden sich nach der verwendeten Modellierungsmethode, der Struktur, der Perspektive und der Methode, wie Verhalten simuliert wird.

### 2.1.2. Partikelsysteme

*Partikelsysteme* modellieren Vorgänge, die auf der Interaktion vieler gleichartiger, häufig kleiner, Teilchen beruhen. Diese **Partikel** können sich gegenseitig anziehen oder abstoßen und bei Berührung Energie verlieren.
Mit Hilfe von Partikelsystemen lässt sich z.B. das Verhalten von Flüssigkeiten oder auch das Verhalten von Personen während einer Evakuierungssituation

simulieren. Letzteres macht Partikelsysteme für uns sehr interessant, da Ergebnisse über andere, gute erforschte Partikelsysteme übertragen werden können.

Bei der Simulation von Partikelsystemen gibt es zwei generelle Ansätze [16]: Beim **mikroskopischen** Ansatz wird versucht, die Bewegungen der Teilchen eines Partikelsystems möglichst exakt vorherzusagen bzw. zu berechnen. Wegen der großen Anzahl an Partikeln ist meistens ein abstrahierender, **makroskopischer** Ansatz notwendig.

Weiterhin gibt es **diskrete** und **stetige** Simulationsansätze. Zur ersten Klasse gehört das sehr eingeschränkte Modell der Zellulären Automaten (siehe Abschnitt 2.1.3). Ein prominentes stetiges Modell ist das **Social Forces Model** [17], bei dem das Verhalten von Personen durch **Kräfte** in Form von Vektoren modelliert wird. Dieses Modell ist erstaunlich mächtig und kann viele komplexe Verhaltensweisen von realen Menschenmengen simulieren.

### 2.1.3. Einführung zelluläre Automaten

**Zelluläre Automaten** (ZA) sind diskrete Modelle in Raum und Zeit. Der Raum ist ein $D$-dimensionales Gitter und die Elemente des Gitters heißen **Zellen**. Jede Zelle befindet sich zu jedem Zeitpunkt in einem Zustand einer endlichen Zustandsmenge. Die Entwicklung einer Zelle wird durch eine **Übergangsfunktion** (deterministisch oder randomisiert) bestimmt und hängt meist vom eigenen Zustand, dem Zustand ihrer Nachbarn und der Zeit ab. Die Übergangsfunktion kann parallel, sequentiell oder in Mischformen angewandt werden. Bei Simulationen von Individuen befindet sich auf einer Zelle entweder kein oder genau ein Individuum.

Die obige Definition ist sehr allgemein und erlaubt viele Freiheiten in der konkreten Modellierung von Fußgängersimulationen. Dazu gehören zum Beispiel die Definition der Nachbarschaft einer Zelle, ihre Größe oder die Anzahl der Zellen, die ein Individuum in einem Zeitschritt benutzt. Bevor eine Bewegung stattfindet muss zunächst das Ziel, zum Beispiel durch die Berechnung eines **statischen Potentials**, bestimmt werden. Hierbei wird in jeder Zelle der Abstand zum Ausgang kodiert. Ein Individuum bewegt sich dann mit größerer Wahrscheinlichkeit zu einer Zelle mit geringerem Abstand. Zusätzlich kann

auch ein **dynamisches Potential** verwenden werden, mit dessen Hilfe Herdeneffekte simuliert werden können.

Einen sehr guten Überblick über Zelluläre Automaten bietet [21].

### 2.1.4. Einführung Agentensysteme

Durch **Agentensysteme** kann das Verhalten von Individuen innerhalb eines Evakuierungsszenarios simuliert werden. Dabei übernehmen **Agenten** die Rolle der Individuen. Agenten sind als autonom anzusehen, besitzen Attribute und können miteinander kommunizieren. In einer Simulation verfolgen Agenten bestimmte Ziele, wie etwa den schnellsten Rettungsweg für sich zu finden. Die Agenten verfolgen dabei Pläne, um ihre Ziele zu erreichen. Diese setzen sich aus unterschiedlichen, einfachen Handlungen zusammen. Da Agenten mehrere Pläne besitzen können, muss eine Vergleichsmöglichkeit für deren Qualität existieren. Für die Bewertung von Plänen werden **Scoring-Funktionen** verwendet, welche Berechnungen auf Grund der Ergebnisse eines Simulationsdurchlaufes durchführen.

Um eine Simulation mit Hilfe eines Agentensystems umzusetzen, benötigt man ein Framework. Dieses lässt sich in zwei Schichten unterteilen. Die **physikalische Schicht** repräsentiert die Umwelt der Agenten. Dort werden die Pläne der Agenten ausgeführt und diese können untereinander kommunizieren. Die **strategische Schicht** repräsentiert das Denken der Agenten. Um die Pläne zu überarbeiten und Entscheidungen zu verbessern, kann hier auf unterschiedliche externe Module zurückgegriffen werden. Zur strategischen Schicht gehört ebenfalls die **Agenten-Datenbank**, in der alle Informationen über die Agenten sowie die Pläne abgespeichert werden.

Für weitere Informationen siehe [27].

## 2.2. Vorträge über Optimierungsthemen

### 2.2.1. Statische Flüsse: Shortest Paths und Maximum Flow

Das **Shortest Paths Problem** gliedert sich in zwei wesentliche Probleme: Beim **Single Source Shortest Path Problem (SSSP)** geht es darum, von einer ausgezeichneten Quelle $s$ die kürzesten Wege zu allen anderen Knoten aus $V$ zu ermitteln, während es beim **All Pairs Shortest Path Problem (APSP)** darum geht, zwischen jedem Knotenpaar einen kürzesten Weg zu ermitteln.
Zur Lösung des **SSSP** wurden der **Algorithmus von Dijkstra** und der **Label Correcting Algorithmus** vorgestellt. Das **APSP** kann man durch jeweils einen SSSP-Aufruf für jeden Knoten lösen. Als Alternative wurde der **Floyd Warshall Algorithmus** präsentiert.

Bei dem **Maximum Flow Problem** (Max Flow Problem) geht es darum, unter Beachtung der Kantenkapazitäten einen möglichst großen Fluss von Quelle $s$ zu Senke $t$ zu schicken.
Der **Algorithmus von Ford & Fulkerson** und der **Capacity Scaling Algorithmus** erreichen dies, indem sie, ausgehend vom Nullfluss, flussvergrößernde Wege suchen und auf ihnen den Fluss erhöhen.
Der **Preflow Push Algorithmus** basiert auf der Idee des Präflusses, bei dem der eingehende Fluss eines Knotens größer als der ausgehende Fluss sein darf, woraus Überschüsse an Knoten resultieren können. Diese Überschüsse sollen weiter in Richtung Senke $t$ getrieben werden, oder, falls dies nicht komplett möglich ist, die verbleibenden Überschüsse zurück Richtung Quelle $s$. Ein guter Überblick über statische Flussprobleme und zugehörige Algorithmen findet sich [2] und [22].

### 2.2.2. Statische Flüsse: Minimum Cost Flows

Beim **Minimum Cost Flow Problem** sind für jede Kante nicht nur Kapazitäten, sondern auch Kosten (pro Flusseinheit) angegeben. Außerdem hat jeder Knoten ein Angebot oder einen Bedarf an Flusseinheiten. Gesucht wird nun der kostengünstigste zulässige Fluss, der diese Angebote/Bedarfe erfüllt.

Die beiden vorgestellten grundlegenden Algorithmen arbeiten jeweils mit einem Fluss, der in Iterationen „besser“ wird.
Beim **Cycle Canceling** Algorithmus ist der Fluss dabei immer zulässig, aber zunächst nicht unbedingt optimal. In jeder Iteration wird der Zielfunktionswert verbessert (Streben nach Optimalität). Der **Successive Shortest Path Algorithmus** (SSP) arbeitet immer mit einem optimalen Fluss und strebt nach Zulässigkeit. Bis diese erreicht ist, sind Überschüsse und Defizite an den Knoten erlaubt.
Beide Algorithmen sind zunächst pseudopolynomiell, können aber so verbessert werden, dass sie in polynomieller Zeit arbeiten (**Minimum Mean Cycle Canceling** Algorithmus bzw. **Scaling**-Varianten des SSP Algorithmus).

Am Ende wurde noch kurz auf den **Network Simplex** Algorithmus eingegangen. Diese netzwerkbasierte Variante der Simplex Methode hat zwar exponentielle Worst-Case-Laufzeit, ist aber der in der Praxis schnellste Algorithmus für das Minimum Cost Flow Problem.

Wie bereits in Abschnitt 2.2.1 erwähnt, enthalten [2] und [22] einen guten Überblick über statische Flussprobleme. Weitere Literatur speziell zum Minimum Cost Flow Problem findet sich z.B. in [6] und [13].

### 2.2.3. LP-Formulierungen für dynamische Flussprobleme

Dynamische Flussprobleme können als **Lineare Programme** aufgefasst werden, bei denen geeignete Zielfunktionen unter bestimmten Nebenbedingungen minimiert bzw. maximiert werden. Dafür wurden im Vortrag **zeitexpandierte Netzwerke** verwendet, die dynamische Flussprobleme auf statische Flussprobleme in einem größeren Netzwerk zurückführen. Nach Einführung dieser Idee wurde sie auf einige dynamische Flussprobleme angewandt.

Um die Linearität der Nebenbedingungen zu sichern, gingen wir von zeitunabhängigen Kapazitäten und konstanten Fahrzeiten aus. Alle Probleme hatten zwei Nebenbedingungen gemeinsam: Zum einen die **Flusserhaltung** und zum anderen die **Einhaltung der Kantenkapazitäten**.

Einige spezielle Probleme erforderten zusätzliche Nebenbedingungen. Beim **Maximum Dynamic Flow Problem** (Dyn Max Flow Problem) z.B. soll der Fluss maximiert werden, der in einem bestimmten Zeitintervall die Supersenke

erreicht. Zur Beschreibung von priorisierten Evakuierungen wurde das **Lexicographical Minimum Cost Dynamic Flow Problem** vorgestellt. Dabei wird der Graph in Prioritätszonen partitioniert und die Aufgabe besteht darin, die Evakuierungszeit unter Berücksichtigung der verschiedenen Prioritäten zu minimieren. Zum Schluss wurde zum einen noch ein Modell mit kontinuierlicher Zeit und zum anderen eines mit flussabhängigen Fahrzeiten vorgestellt.

Für weitere Einblicke in die betrachteten Problemstellungen sei hier auf [16] verwiesen.

### 2.2.4. Lösungsansätze dynamischer Flüsse

**Grundlagen & Dyn Max Flow** Es wurden die Zerlegungen von statischen Flüssen in **Flussstränge (Chain Decomposition)** und die Algorithmen, die diese Zerlegung vornehmen vorgestellt, da sie eine wichtige Grundlage für die Arbeit mit dynamischen Flüssen darstellen. Das **Dyn Flow Problem** (Maximale Menge Fluss soll in Zeit $T$ von Quellenmenge $S^+$ aus die Senkenmenge $S^-$ erreichen) wurde vorgestellt und die Ford und Fulkerson'sche Reduktion auf **Minimum Cost Circulation** (Finden einer kostengünstigsten maximalen Zirkulation im Netzwerk) wurde erläutert.

**Universally Dyn Max Flow Problem** Das **Universally Dyn Max Flow Problem** wurde als das Problem eingeführt, in einem Netzwerk sowohl die Flussmenge, die aus der Quelle pro Zeiteinheit austritt, als auch die Flußmenge, die pro Zeiteinheit bei der Senke eintrifft, zu maximieren. Es wurde ein Algorithmus von Hoppe vorgestellt, der dieses Problem für den Fall einer Quelle und einer Senke löst. Da dieser jedoch im Worst-Case exponentielle Laufzeit hat, wurde danach noch eine $1/(1+\varepsilon)$-Approximation für das Problem präsentiert, die durch geschicktes Runden der Eingabe eine polynomielle Laufzeit erzielt.

Der grundlegende Algorithmus von Ford und Fulkerson wurde im Original in [12] beschrieben, der Vortrag orientiert sich aber mehr an der Darstellung der Algorithmen in [18].

### 2.2.5. Lexicographically Maximum Dynamic Flows

**Lexikographisch Maximale Flüsse** Das **Dyn Max Flow Problem** lässt sich auf mehrere **Terminale** (Quellen und Senken) erweitern, indem zusätzlich eine Ordnung auf den Terminalen definiert wird. Gesucht ist dann ein Fluss, der bzgl. dieser Ordnung **lexikographisch** maximal ist. Das so erweiterte Problem heißt **lexikographisch maximales Flussproblem** (Lex Max Problem). Analog lassen sich Earliest-Arrival-Flüsse erweitern. Beide Probleme wurden im Vortrag eingeführt. Sie finden z.B. Anwendung in einer Evakuierung mit prioritisierten Zonen.

**Algorithmus von Hoppe** Im statischen Fall existiert für das Lex Max Problem ein einfacher Algorithmus von Minieka, den Hoppe 1995 auf den dynamischen Fall übertragen konnte. Der Algorithmus verwendet *kein* zeitexpandiertes Netzwerk, sondern benötigt lediglich für jedes Terminal eine Berechnung eines statischen **Minimalkostenflusses** und läuft daher in Polynomialzeit. Dieser Algorithmus wurde im Vortrag vorgestellt.

Der Algorithmus basiert auf **verallgemeinerten zeitlich wiederholten Flüssen**, welche im Vortrag kurz eingeführt wurden.

Die vorgestellten Algorithmen werden in [26], [18] und [28] beschrieben.

### 2.2.6. Earliest Arrival Transshipment

**Das Quickest Transshipment Problem** Die Berechnung maximaler dynamischer Flüsse und auch lexikographisch maximaler Flüsse maximiert für einen vorgegebenen Zeithorizont die Flussmenge, die die Quelle(n) verlässt. Bei der Evakuierung eines Gebäudes wissen wir aber unter Umständen nicht, wie viel Zeit bleibt, bis das Gebäude einstürzt. Statt möglichst viele Personen in einem vorgegebenen Zeitraum zu evakuieren, möchten wir daher viel lieber die Zeit minimieren, die man benötigt, um alle Personen zu evakuieren, die sich gerade im Gebäude aufhalten. Dafür müssen wir zusätzlich zum Netzwerk angeben, wie viele Personen sich in der Quelle / den Quellen befinden. Auf diese Weise erhalten wir das **Quickest Flow Problem** und das **Quickest Transshipment**

**Problem**. Der Vortrag beschäftigte sich mit der Rückführung dieser Probleme auf das **Dynamic Transshipment Problem**. Diese Reduktion wurde von Hoppe [18] entwickelt.

**Das Earliest Arrival Transshipment Problem** Wenn wir ein brennendes Gebäude evakuieren, möchten wir vielleicht nicht nur erreichen, dass das Gebäude möglichst schnell leer ist, sondern auch, dass zu jedem früheren Zeitpunkt so wenig Personen im Gebäude sind wie möglich. Mit dieser Fragestellung beschäftigt sich das **Earliest Arrival Transshipment Problem**, das in der zweiten Hälfte des Vortrags behandelt wurde. Der dazu vorgestellte Algorithmus findet sich in [5].

### 2.2.7. Probleme mit zeit- und flussabhängigen Fahrzeiten

Netzwerkflussmodelle, die auf konstante Fahrzeiten beschränkt sind, können keine dynamischen Abläufe modellieren. Hier schaffen Netzwerkflussmodelle mit zeit- und flussabhängigen Fahrzeiten Abhilfe. Im Vortrag wurden solche Modelle beschrieben, wobei von diskret modellierter Zeit ausgegangen wurde.

**Zeitabhängige Fahrzeiten** In diesem Fall werden die Fahrzeiten durch eine Funktion $t : E \times \{0, \dots, T\} \to \mathbb{N}_0$ beschrieben, wobei $E$ die Kantenmenge des Graphen und $T$ der Zeithorizont des Problems ist. Dadurch werden Phänomene wie sich ausbreitendes Feuer modellierbar.
Aufgrund des diskreten Zeitmodells muss man sicherstellen, dass die Zeitspanne, in der sich Fluss auf einer Kante befindet, immer ganzzahlig ist. Dies kann man z.B. mit dem **Frozen Arc Model** und dem **Elastic Arc Model** erreichen. Detailierte Ausführungen sind unter anderem bei Tjandra [32] zu finden.

**Flussabhängige Fahrzeiten** Hier werden die Fahrzeiten durch eine Funktion $t : E \times \mathit{Fluss} \to \mathbb{N}_0$ beschrieben, wobei $E$ die Kantenmenge des Graphen ist und *Fluss* ein Maß für den Fluss auf der Kante. Es gibt dabei unterschiedliche Möglichkeiten, den Fluss auf einer Kante zu messen, z.B. das **Inflow Dependent** und das **Load Dependent** Fahrzeitmodell.

Ausführliche Erläuterungen zu diesen Modellen finden sich etwa in Baumann und Köhler [4], Hall, Langkau und Skutella [15], Köhler und Skutella [20] und Langkau [25].

### 2.2.8. Erweiterte zeitexpandierte Graphen

Erweiterte zeitexpandierte Graphen werden benötigt, um flussabhängige Fahrzeiten zu modellieren. Wie bereits in Abschnitt 2.2.7 erwähnt, lässt sich die Fahrzeit in solchen Fällen als $t : E \times \text{Fluss} \rightarrow \mathbb{N}_0$ darstellen. Um eine diskrete Fahrzeitfunktion modellieren zu können, wird von einer konvexen Funktion ausgegangen, die zwischen den ganzzahligen Stützpunkten stückweise linear gemacht wird.

**Der Ansatz von Carry und Subrahmanian:** Bei dem Ansatz von Carry und Subrahmanian [7] werden die verschiedenen Fahrzeiten im zeitexpandierten Netzwerk durch zusätzliche Kanten zwischen den Zeitschichten modelliert. Die Kapazität einer solchen Kante, die z.B. $v_i$ und $w_{i+t}$ verbindet, ist gleich dem maximalen Fluss, der die Kante in $t$ Zeiteinheiten durchqueren kann. Der ganze Graph wird dann als ein lineares Programm aufgefasst.

**Der Ansatz von Köhler, Langkau und Skutella:** Dieser Ansatz geht von einer stückweise konstanten Fahrzeitfunktion aus und modelliert das Problem mittels eines sog. Bogengraphen so, dass übliche Netzwerkalgorithmen anwendbar sind. Er wird in [19] beschrieben.

# 3. Organisation der Arbeit

## 3.1. Ablauf des 1. Semesters

Nach der Seminarphase stand für uns eine Planungsphase (05.11.-19.11.2007) an, um drängende Fragen zu klären: Was für Ziele haben wir genau? Welche einzelnen Aufgaben gibt es, und in welcher Reihenfolge müssen wir sie bewältigen? Wie lange werden die einzelnen Schritte dauern?

In der nachfolgenden inhaltlichen Diskussion haben wir die Ziele für das erste Semester gesteckt und einzelne Aufgaben ausgemacht. Dabei haben wir das Schnittstellendiagramm in Abbildung 3.1 entwickelt. Anschließend haben wir diese in Phasen gegliedert und für die einzelnen Phasen den Zeitaufwand geschätzt. Dabei haben wir jede Phase sowie die dafür vorgesehene Zeit in einen Konzeptions- und einen Implementierungsteil aufgeteilt.

Einen ähnlichen Plan haben wir am Ende des ersten Semesters für das zweite Semester aufgestellt.

Im Folgenden listen wir auf, welche Phasen wir durchgeführt haben und was während dieser erledigt wurde. Dabei berichten wir den tatsächlichen Ablauf. Bei der ursprünglichen Planung konnten wir den Zeitaufwand für manche Teilschritte nicht exakt schätzen, so dass wir den Plan später etwas verändert haben. Diesen angepassten Ablauf stellen wir jetzt vor.

### 3.1.1. Phase 1

Für die erste Phase hatten wir zwei Wochen vorgesehen, d.h. den Zeitraum vom 19.11.2007 bis zum 03.12.2007. Diesen haben wir auch eingehalten.

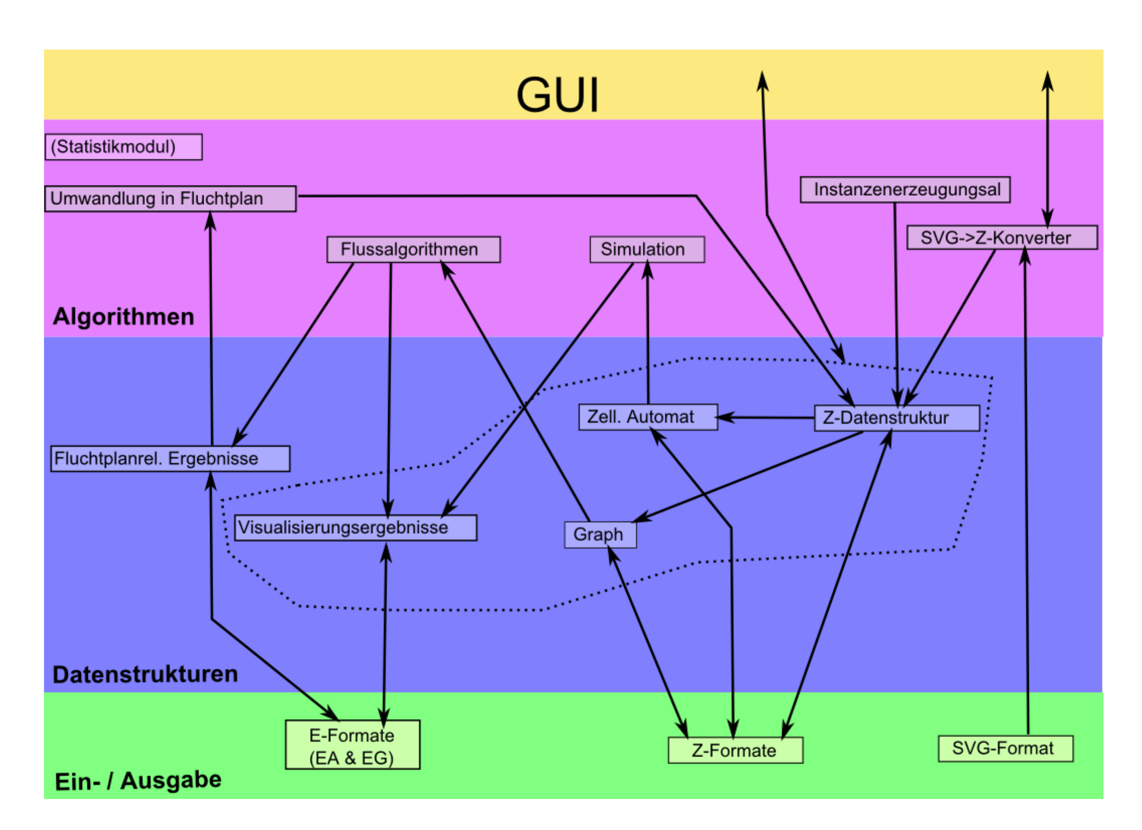



**Abbildung 3.1.:** Ein Schnittstellendiagramm als Übersicht über die geplanten Komponenten unseres Programms.

In dieser Phase ging es um die Entwicklung der wichtigsten Datenstrukturen. Zur Darstellung von Gebäuden in unserem Programm haben wir das *Z-Format* bzw. die beiden grundlegenden Komponenten *Z-Plan* und *Z-Belegung* entwickelt und implementiert. Außerdem haben wir die Datenstrukturen für den Graphen (*Z-Graph*) sowie den Zellulären Automaten (*Z-Zellulärer Automat*) als weitere Komponenten des Z-Formats entworfen und implementiert. Zur Sammlung der Daten und Speicherung auf der Festplatte haben wir dabei noch das *Z-Projekt* entwickelt.

Die folgende Gruppenaufteilung galt sowohl für Entwurf als auch für die Implementierung:

| Gruppenaufteilung: | Gruppenmitglieder |
|---|---|
| Graphen | **Martin**, Melanie, Moukarram |
| Zellulärer Automat | Daniel, Marcel, **Matthias**, Sophia |
| Z-Format | **Jan**, Joscha, Sylvie, Gordon, Timon |

Die Verantwortlichen für die Teilprojekte sind in Fettschrift markiert.

### 3.1.2. Phase 2a

Der Name dieser Phase ist darin begründet, dass sie in unserem ursprünglichen Plan nur den ersten Teil von Phase 2 dargestellt hat. Wegen unvorhergesehener Schwierigkeiten hat sich die Umsetzung dieser Phase aber verzögert, so dass wir Phase zwei in zwei Teilphasen aufgeteilt haben. Wir haben an Phase 2a vier Wochen gearbeitet, d.h. vom 03.12.2007 bis zum 14.01.2008.

Dabei wurden die Datenstrukturen weiterverwertet, die wir in der ersten Phase entwickelt hatten. Dies bedeutet zum einen, dass zwei Umwandlungen entworfen wurden, mit deren Hilfe man aus Z-Plänen und Z-Belegungen Graphen und Zellulären Automaten gewinnen kann. Außerdem haben wir die Phase verwendet, um eine Rasterung der im Z-Format vorliegenden Daten durchzuführen, um die Umwandlungen zu unterstützen. Die Notwendigkeit, eine Rasterung zu implementieren, war der Grund für die entstandene Verzögerung.

Der andere große Bereich dieser Phase war der Entwurf eines Editors für das Z-Format, in dem Gebäudepläne gezeichnet und gespeichert werden können.

Die Gruppen waren während dieser Phase folgendermaßen eingeteilt:

| Gruppenaufteilung: | Gruppenmitglieder |
|---|---|
| Rasterung | gruppenübergreifend |
| Graphen | Joscha, **Martin**, Melanie, Moukarram |
| Zellulärer Automat | Daniel, Marcel, **Matthias**, Sophia, Sylvie |
| Z-Editor | Jan, Gordon, **Timon** |

### 3.1.3. Phase 2b

Durch die Verzögerung von Phase 2a haben wir den Rest des ersten Semesters für Phase 2b vorgesehen, d.h. diese Phase nahm die drei Wochen vom 14.01.2008 bis zum 04.02.2008 ein.

Phase 2b beschäftigte sich nun mit den Algorithmen: Zum einen sollten zwei Flussalgorithmen ausgewählt und umgesetzt werden, zum anderen musste das Verhalten des zellulären Automaten spezifiziert werden.

Bei den Flussalgorithmen mussten wir die Vor- und Nachteile der einzelnen Algorithmen abwägen, um eine sinnvolle Wahl aus der großen Menge möglicher Algorithmen zu treffen. Beim Entwurf des Zellulären Automaten haben wir uns hauptsächlich an den Ideen aus dem in Abschnitt 2.1.3 beschriebenen Vortrag orientiert. Diese mussten natürlich zuerst daraufhin überprüft werden, ob sie sich für Evakuierungsszenarien wirklich eignen. Anschließend musste eine sinnvolle und allgemein gehaltene Strukturierung entwickelt werden.

Wir haben uns folgendermaßen auf die einzelnen Gruppen verteilt:

| Gruppenaufteilung: | Gruppenmitglieder |
|---|---|
| Flussalgorithmen | Gordon, Joscha, **Martin**, Moukarram, Melanie, Timon |
| Zellulärer Automat | Daniel, Jan, Marcel, **Matthias**, Sophia, Sylvie |

## 3.2. Ablauf des 2. Semesters

Zu Beginn des 2. Semesters haben wir zunächst Bilanz gezogen, neue Ziele gesteckt und erneut einen Phasenplan für die kommenden Monate aufgestellt. Außerdem haben wir eine Testevakuierung des Audimax' der TU Dortmund geplant und am 22.04.2008 im Anschluss an die Vorlesung „Datenstrukturen, Algorithmen und Programmierung II“ durchgeführt.

Unser neuer Phasenplan begann direkt im Anschluss an dieses Event und ist im Folgenden aufgelistet. Im zweiten Semester war die Zusammenarbeit zwischen uns schon gut eingespielt, so dass wir ohne Verantwortliche in den Gruppen ausgekommen sind und daher auf diese verzichtet haben.

### 3.2.1. 3. Phase

Zu Beginn des 2. Semesters haben wir vom 22.4. bis zum 29.4. eine kurze Phase eingelegt, um Bugs zu beheben, den Quellcode besser zu kommentieren und den Editor zu erweitern. Außerdem haben wir eine Planungsgruppe für das Aussehen der GUI und der Visualisierung eingerichtet, um einen reibungslosen Übergang zur anschließenden Visualisierungsphase zu schaffen. Neben der

Benutzerführung und der Darstellung von Gebäuden, Graphen und Zellulären Automaten hat sich diese Gruppe auch Gedanken über das Dateimanagement gemacht. Wir haben uns dabei folgendermaßen aufgeteilt:

| Gruppenaufteilung: | Gruppenmitglieder |
|---|---|
| Graph | Gordon, Martin, Melanie, Moukarram, Timon |
| Zellulärer Automat | Daniel, Marcel, Sophia, Sylvie |
| Editor | Jan |
| GUI und Visualisierung | Jan, Matthias, Joscha |

### 3.2.2. 4. Phase

In der zweiten bis fünften Woche (29.4. bis 27.5.) haben wir uns dann wieder in neue Gruppen aufgeteilt. Die größte Gruppe war dabei die Visualisierungsgruppe, die sich mit der Umsetzung der geplanten Visualisierung mit OpenGL beschäftigt hat. Dabei ging es zunächst um das grundlegende Gerüst, die Vervollständigung der Visualisierung hat auch in den weiteren Phasen noch Zeit beansprucht.

Parallel dazu wurde die statistische Auswertung der Simulation und der Flussalgorithmen durch eine weitere Gruppe geplant und implementiert. Für diese umfangreiche Aufgabe war eine Gruppe nötig, die sich sowohl mit unserer Umsetzung der Graphalgorithmen als auch mit dem Zellulären Automaten auskennt.

Die dritte neue Gruppe hat sich während dieser Zeit damit beschäftigt, wie Treppen in das Programm eingebaut werden können. Dabei waren alle Datenstrukturen betroffen, d.h. die Ergänzung musste im Z-Format, den Graphdatenstrukturen und im Zellulären Automaten überlegt und umgesetzt werden.

Wir haben uns folgendermaßen auf die Gruppen verteilt:

| Gruppenaufteilung: | Gruppenmitglieder |
|---|---|
| Visualisierung | Daniel, Gordon, Jan, Joscha, Marcel, Moukarram |
| Statistik | Martin, Matthias, Sylvie |
| Treppen | Melanie, Sophia, Timon |

### 3.2.3. 5. Phase

In der nächsten Phase (27.5.-10.6.) ging es um die Weiterentwicklung und Vervollständigung der Visualisierung und der Statistiken. Die Ergänzung um Treppen war abgeschlossen, so dass sich diese Mitglieder auf die anderen Gruppen verteilen konnten.

### 3.2.4. 6. Phase

In der Zeit vom 10.6. bis zum 26.6. haben wir vier Gruppen gebildet. Zwei der Gruppen waren dabei die Weiterführungen der Visualisierungs- und Statistikgruppen mit eingeschränkter Mitgliederzahl. Außerdem haben wir uns in dieser Zeit damit befasst, wie man unser Z-Format in das allgemein genutzte AutoCAD-Format dxf exportieren kann. Die große neue Gruppe in dieser Phase war die Kalibrierungsgruppe. Diese hat sich damit beschäftigt, das Verhalten des Zellulären Automaten z.B. anhand der standardisierten Rimea-Tests zu kalibrieren. Die Gruppenaufteilung sah folgendermaßen aus:

| Gruppenaufteilung: | Gruppenmitglieder |
|---|---|
| Visualisierung | Daniel, Jan, Melanie, Moukarram |
| Statistik | Martin, Matthias, Timon, Sylvie |
| Kalibrierung | Gordon, Joscha, Marcel, Moukarram, Sophia |

### 3.2.5. 7. Phase

Die letzte neue Gruppe kam dann in der Zeit vom 26.6. bis zum 8.7. hinzu. Neben der weiterlaufenden Gruppe für Kalibrierung haben wir eine Gruppe für Fluchtpläne gebildet, die eine Verbindung zwischen dem Graphen und den Zellulären Automaten geschaffen hat. Außerdem wurde ein zusätzlicher Graphalgorithmus implementiert. Wir haben in folgenden Gruppen gearbeitet:

| Gruppenaufteilung: | Gruppenmitglieder |
|---|---|
| Kalibrierung | Gordon, Joscha, Marcel, Moukarram, Sophia |
| Fluchtpläne | Daniel, Jan, Martin, Matthias, Melanie, Sylvie, Timon |

### 3.2.6. 8. Phase

Vom 8.7. bis zum 18.7. haben wir gemeinsam den Endbericht geplant und die Vervollständigung und Verwendung des Programms diskutiert. Außerdem wurden Fehler behoben und noch fehlende Features implementiert. Dazu haben wir keine Gruppenaufteilung mehr vorgenommen.

# Teil II.

# Konzept und Implementierung

# Inhaltsverzeichnis

# 4. Einleitung

Das Ziel unserer Projektgruppe war der Entwurf eines Tools zur Evakuierung von Gebäuden. Dabei sollten Simulation und exakte Flussalgorithmen verwendet werden. Im Bereich der Simulation haben wir uns entschieden, ausschließlich mit Zellulären Automaten zu arbeiten und diese unseren Wünschen entsprechend zu erweitern. Evakuierungsszenarien lassen sich mit Zellulären Automaten hinreichend gut beschreiben. Als weiteren interessanten Ansatz haben wir das in Abschnitt 2.1.2 beschriebene Social Forces Modell empfunden, eine Implementierung dieser Idee wäre jedoch innerhalb der gegebenen Zeit zu umfangreich gewesen. Im Bereich der Flussalgorithmen haben wir verschiedene Algorithmen implementiert, die weiter unten beschrieben werden.

## 4.1. Aufbau unseres Programms

Der erste wichtige Teil unseres Programmpakets ist der **Z-Editor**. Mit diesem Editor kann der Benutzer der Software sein Gebäude zeichnen, wobei das „Abzeichnen“ von vorhandenen Gebäudeplänen durch transparentes Einblenden unterstützt wird.

Programmintern wird ein Gebäudeplan dann in einem Zwischenformat gespeichert, dass wir **Z-Format** nennen. Die beiden wichtigsten Komponenten dieses Formats sind der **Z-Plan** für den eigentlichen Grundriss und **Z-Belegungen**, mit denen verschiedene Möglichkeiten beschrieben werden können, wo sich im Gebäude vor der Evakuierung wie viele Personen aufhalten.

Außerdem können wir unser Z-Format in andere Datenstrukturen umwandeln, genauer gesagt ist die Erzeugung von Graphen und Zellulären Automaten möglich. Dabei ist die Umwandlung so gestaltet, dass die erzeugten Datenstrukturen möglichst ähnlich sind. Wenn bei der Umwandlung Abweichungen von

der Realität entstehen, so fließen diese möglichst gleichermaßen in beide Datenstrukturen ein. Damit soll die Vergleichbarkeit der beiden Modelle sichergestellt werden. Wir verwalten die Ergebnisse als **Z-Graph** und **Z-Zellulärer Automat**.

Für die Organisation mehrerer Simulationsdurchläufe haben wir einen Batch-Dialog entwickelt, in dem der Benutzer die gewünschten Parameter einstellen kann.

Im Anschluss an die virtuelle Durchführung der Evakuierung kann man sich dann die Ergebnisse zum einen in Form von Statistiken und zum anderen als Visualisierung ansehen.

Im Folgenden beschreiben wir die verwendeten Datenstrukturen und Algorithmen genauer. Die Statistik und Visualisierung wird hier nicht beschrieben, auf die Benutzung dieser Programmteile gehen wir im Benutzerhandbuch ein (Teil IV).

# 5. Datenstrukturen und Dateistruktur

## 5.1. Das Z-Format

Ein Evakuierungsszenario wird intern im **Z-Format** gespeichert. Darunter wird sowohl die interne Z-Datenstruktur verstanden, als auch das zugehörige Dateiformat zum Speichern eines **Z-Projektes**. Die Gebäude- und Szenariodaten werden zentral in einem **Z-Projekt** gespeichert, während die Ergebnisse aus den Simulationen bzw. Graphalgorithmen in Form von **BatchResults** festgehalten werden (siehe Abschnitt 5.4).

Das Z-Format bietet die Möglichkeit, einen Gebäudeplan (**Z-Plan**) zu speichern und diesen durch Zonen, die besondere Gebiete kennzeichnen, zu ergänzen. Außerdem werden Informationen über zu evakuierende Personen in **Z-Belegungen** gespeichert. Aus diesen (editierbaren) Daten können anschließend geeignete Datenstrukturen für Graphalgorithmen und Zelluläre Automaten erzeugt werden, die als **Z-Graph** und **Z-Zellulärer Automat** (nur) zusammen mit den jeweiligen Ergebnissen im BatchResult gespeichert werden (BatchResults werden am Ende dieses Kapitels beschrieben).

Physikalisch werden die Informationen aus dem Z-Projekt und aus dem BatchResult in jeweils einer einzelnen Datei gespeichert, wobei allerdings die BatchResult-Dateien auch das Z-Projekt enthalten. Dies bedeutet zwar, das Daten doppelt abgelegt werden, aber es vermeidet auch das Problem, die BatchResult-Datei und die Z-Projekt-Datei miteinander synchronisieren zu müssen und es vereinfacht das Einlesen der Daten. Die Dateien werden als XML-Datei mittels XStream gespeichert, dadurch entfällt die Entwicklung eines eigenen Dateiformates. Zur effizienten Speicherung wurden jedoch für die einzelnen Objekte

die Laderoutinen angepasst. Für die Z-Projekte wurde die Dateiendung .zet und für die BatchResults die Dateiendung .ers vorgesehen.

Im Folgenden werden die einzelnen Komponenten des Formats genauer erläutert.

### 5.1.1. Das Z-Projekt

Ein Z-Projekt wird hierarchisch gespeichert, beginnend mit der Hauptklasse `Project`. Diese fungiert als Container für die anderen Projektinhalte. Ein Objekt der Klasse `Project` enthält genau einen Gebäudeplan, der von der Klasse `BuildingPlan` gekapselt wird und eine Liste der gültigen Belegungen, dargestellt durch die Klasse `Assignment`.

Es ist immer jeweils eine Belegung als aktiv ausgewählt (es sei denn, es gibt keine Belegungen, dann ist auch keine ausgewählt). Mit der Methode `setCurrentAssignment( Assignment )` kann diese entsprechend gesetzt werden. Es werden dann automatisch nur die `AssignmentAreas` (Belegungsbereiche) im Plan angezeigt, die auch zu der aktuellen Belegung gehören.

Geladen und gespeichert wird ein Projekt mit den statischen Methoden `load( String )` und `save()` beziehungweise `save( String )`. Das Projekt wird dabei mit XStream gespeichert, die Klassenmethode gibt dann ein neues Objekt zurück, das den Daten aus der XML-Datei entspricht.

### 5.1.2. Der Z-Plan

Ein Z-Plan besteht im Wesentlichen aus einer Liste von **Stockwerken**. Die Stockwerke selbst können Räume enthalten. Diese müssen sich immer ganz in einem Stockwerk befinden und sind zweidimensional.

Die Räume innerhalb einer Etage sowie die enthaltenen ausgezeichneten Gebiete werden als Polygone modelliert. Diese Polygone müssen nicht konvex sein, dürfen jedoch keine Selbstschnitte oder „Löcher“ enthalten. Dadurch wird ein effizientes Speichern und Bearbeiten der Objekte ermöglicht, während gleichzeitig die meisten vorkommenden Räume leicht modelliert werden können.

### Polygone

Die generische Klasse `PlanPolygon` wird zur Darstellung von schnitt- und lochfreien Polygonen verwendet. Ein solches Polygon begrenzt immer eine Fläche.

Ein Polygon wird als doppelt verkettete Liste von Kanten des Typs `Edge` bzw. den zugehörigen Punkten dargestellt. Auch die Klasse `Edge` ist generisch. Die Koordinaten der Eckpunkte werden als Ganzzahlen angegeben. Das diskrete Raster hat eine Schrittweite von Millimetern, so dass eine ausreichend genaue Angabe von Koordinaten gewährleistet ist. Die diskreten Koordinaten verringern Fehler bei der Konsistenzprüfung.

Rundungen (z.B. runde Wände) werden nicht explizit unterstützt und müssen über eine Folge von Punkten approximiert werden.

Die Klasse `PlanPolygon` kapselt Funktionen, die zum Verwalten der eingeschlossenen Fläche notwendig sind. Über `getEdges()` können sämtliche zu einem Polygon gehörende Kanten abgefragt werden. Es sind viele geometrische Funktionen vorhanden, z.B. um zu prüfen, ob ein Polygon in einem anderen komplett enthalten ist ober ob Polygone disjunkt sind. Außerdem kann getestet werden, ob sich Kanten zweier Polygone schneiden und ob ein Punkt in einem Polygon enthalten ist. `PlanPolygon`-Objekte stellen auch Iteratoren bereit, mit denen durch die Punkte oder Kanten des Polygons iteriert werden kann.

### Etagen und Räume

Eine Etage wird durch einen eindeutigen Namen identifiziert und durch den Typ `Floor` repräsentiert. Sie ist im Wesentlichen nur ein Container und enthält eine Liste sämtlicher Räume dieser Etage.

Die Räume werden durch `Room`-Objekte modelliert und haben einen Bezeichner, durch den ein Benutzer verschiedene Räume identifizieren kann. Die Klasse `Room` erbt direkt von der Klasse `PlanPolygon` und verwendet als Kanten die von `Edge` abgeleitete Klasse `RoomEdge`. `RoomEdge` erweitert die Kanten so, dass sie verbunden werden können, um einen Durchgang zu erzeugen, d.h. so, dass sie durchlässig sein können. Dies dient der Modellierung von Türen.

Durch die Kantenbegrenzung ergibt sich der Rauminnenumriss im Zweidimensionalen, also die im Programm für die Evakuierung real nutzbare Fläche.

### Bereiche / Zonen

Räume können besondere Bereiche enthalten, die von der generischen, abstrakten Klasse `Area` und indirekt von `PlanPolygon` erben, jedoch werden die Kanten der Bereiche von allgemeinen Kanten des Typs `Edge` dargestellt. Ein Bereich muss vollständig in dem Raum enthalten sein, zu dem er gehört. Folgende Gebietstypen werden unterstützt:

- Verzögerungsbereiche (Klasse `DelayArea`) beschreiben Gebiete, in denen die Bewegungsgeschwindigkeit reduziert ist. Die Geschwindigkeitsänderung wird dabei als Wert aus dem Intervall $]0, 1]$ über die Methode `setSpeedFactor()` angegeben und stellt den Prozentsatz der Originalgeschwindigkeit dar. Bei überlappenden Verzögerungsbereichen gilt im Schnittbereich das Produkt der überlappenden Reduktionsfaktoren.
- Treppenbereiche (Klasse `StairArea`) beschreiben die Eigenschaft von Treppen, dass man zwar die Treppe in beiden Richtungen nicht so schnell durchlaufen kann wie eine gerade Fläche, dass man aber trotzdem schneller runter als rauf laufen kann. Als unten oder oben muss man dabei jeweils einen zusammenhängenden Teil der Polygonkanten angeben. Die Teile dürfen sich natürlich nicht überschneiden. Die Geschwindigkeitsänderung beim Laufen von oben nach unten und andersrum ist dabei einzeln als Wert aus dem Intervall $]0, 1]$ anzugeben (analog zu Verzögerungsbereichen).
- Nicht betretbare Bereiche werden von der Klasse `InaccessibleArea` dargestellt und sind Gebiete, die nicht betreten werden können.
- Evakuierungsbereiche (Klasse `EvacuationArea`) sind die Bereiche, die Personen erreichen müssen, um als evakuiert zu gelten. Sobald eine Person einen solchen Bereich erreicht, wird sie aus dem Programmablauf entfernt.
- Sichere Bereiche (Klasse `SaveArea`) stellen Gebiete dar, die als sicher gelten. Sie werden vor allem für die Simulation verwendet. Personen, die

sich in einem sicheren Bereich aufhalten, sind genauso gerettet wie Personen, die schon einen Evakuierungsbereich erreicht haben, werden aber noch nicht aus der Simulation entfernt. Damit kann simuliert werden, dass sich auch *vor* Gebäuden noch Engpässe bilden können. Eine Evakuierung ist dann vollständig, wenn alle Personen evakuiert sind, d.h. entweder Evakuierungsbereiche erreicht haben (und damit aus der Simulation entfernt wurden) oder sich in sicheren Bereichen aufhalten. Im Falle von Überlappungen mit sicheren Bereichen gilt, dass Evakuierungsbereiche höhere Priorität haben. Weiteres zu Evakuierungs- und sicheren Bereichen in Abschnitt 7.1.5.

- Belegungsbereiche (Klasse `AssignmentArea`) kennzeichnen Bereiche, in denen sich (vor der Evakuierung) Personen aufhalten können. Jedem Belegungsbereich wird eine bestimmte Anzahl von Personen sowie die Art der Personen zugeordnet. Durch verschiedene überlappende Belegungsbereiche können inhomogene Personengruppen dargestellt werden. Näheres dazu im Abschnitt 5.1.3.

- Barrieren (Klasse `Barrier`) stellen keine Gebiete im eigentlichen Sinne dar, sondern sind nur ein offener Polygonzug, der Wände darstellt. Diese Wände sollen schmale Hindernisse im Inneren von Räumen darstellen. Da viele unserer Polygonoperationen aber nur auf geschlossenen Polygonen korrekt arbeiten, sind diese Polygone intern auch geschlossene Polygone, die aber immer Fläche 0 haben, so dass sie wie offene Polygone wirken.

Ein `Vector` mit den entsprechenden Bereichen kann über die Methoden `getSaveAreas()`, `getDelayAreas()`, `getAssignmentAreas()` und `getBarriers()` abgerufen werden. Neue Gebiete werden mittels `addArea( Area area )` hinzugefügt. Durch die abstrakte Basisklasse können hier beliebige Gebiete hinzugefügt werden.

#### Wände und Türen

Im Z-Plan werden Wände implizit modelliert: Durch die Polygonzüge werden die Innenmaße eines Raumes dargestellt. Dadurch sind die Abstände zwischen den Innenumrissen zweier aneinander angrenzender Räume implizit Wände. Um die Räume zu verbinden, zwischen denen ein solcher Abstand ist, benötigt

man **Türräume**, d.h. kleine Räume, die den Durchgang zwischen zwei Räumen modellieren und an den beiden Enden jeweils durch durchlässigen Kanten mit den beiden Räumen verbunden sind. Falls dies nicht gewünscht ist, können Räume auch direkt verbunden werden, ohne dass der Plan Wände enthält. Für die Personen sind die Wände dann nicht vorhanden. In diesem Fall muss man die Räume so modellieren, dass sich ihre Innenumrisse berühren.

Als Passagen zwischen den Räumen können nur Kanten von Polygonen benutzt werden. Wenn zwei Räume miteinander verbunden werden sollen, müssen sich die Verbindungskanten genau übereinander befinden und die gleiche Länge haben.

Übergänge zwischen Stockwerken werden ebenfalls durch Kanten realisiert. Dazu müssen einfach zwei Kanten gleicher Länge auf zwei Stockwerken miteinander verbunden werden (bzw. als verbunden markiert werden). In diesem Fall ist die Position egal, Durchgänge können also auch an anderen Koordinaten auf verschiedenen Stockwerken existieren. Personen können so ohne Zeitaufwand von einer Etage in eine andere wechseln. Deshalb werden solche Kanten Teleport-Kanten genannt. Um für Stockwerkwechsel Zeitverluste einzubauen, benötigt man (analog zu den Türräumen) zusätzliche Räume, die z.B. in einem zusätzlichen (virtuellen) Stockwerk liegen. Dieses wird im nächsten Abschnitt im Zusammenhang mit Treppen noch einmal aufgegriffen.

Es ist nicht vorgesehen, durch Teleport-Kanten (Klasse `TeleportEdge`) zwei Räume auf demselben Stockwerk zu verbinden, obwohl das Modell dieses Verhalten theoretisch unterstützen würde.

### Treppen

Treppen müssen als `StairArea` innerhalb eines Raumes modelliert werden. Da eine Person, die eine solche „Treppe“ benutzt, zunächst formal in derselben Etage verbleibt (denn die StairArea modelliert nur die Verzögerung durch die Treppenbenutzung) muss der Übergang zur nächsten Etage gesondert modelliert werden, z.B. durch eine direkt an die StairArea anschließende Teleport-Kante.

Als Beispiel folgen zwei Modellierungsbeispiele:

**Audimax der TU-Dortmund** Die langgezogene Treppe des Mittelgangs wird als `StairArea` innerhalb eines großen Audimax-Raums dargestellt. Am Ende befindet sich eine `TeleportEdge`, die in das Foyer führt (ohne Zeitverzögerung). Der gesamte Raum befindet sich im Modell im Erdgeschoss, die oberen Türen führen jedoch in die erste Etage.

**Wendeltreppe** Die Treppe wird mit einer einleitenden `TeleportEdge` am Treppenanfang begonnen. Dann folgt eine neu einzufügende, virtuelle Etage, die **Treppenetage**. In dieser Etage wird die Wendeltreppe **linearisiert** dargestellt, also als Flur mit einer gewissen Länge und (geringer) Breite, auf der man dann eine `StairArea` platzieren sollte, um auch die Verzögerung durch die Treppe korrekt abzubilden. Diesen Flur müssen die Personen durchlaufen. Hier können sich auch Staus bilden, so dass die Verzögerung durch die Treppe dargestellt werden kann. Von dieser **Treppenetage** aus führt dann ein zweites Paar von `TeleportEdges` zur Zieletage.

Auch andere Konstellationen sind denkbar, da hier wirklich eine neue Etage eingeführt werden mus, und man somit beim Entwerfen der virtuellen **Treppenetage** völlig freie Hand hat. Zum Beispiel kann man sie auch mit verschiedenen `StairAreas` belegen. Auch langgezogene Treppenhäuser (Beispiel Parkhaus) mit mehreren Ausgängen zu verschiedenen Etagen sind durch passende **linearisierte** Anordnungen simulierbar.

### 5.1.3. Die Z-Belegung

Ein Z-Projekt unterstützt verschiedene **Belegungen**, d.h. bestimmte Ausgangssituationen für die Evakuierung. Bei der Modellierung eines Hörsaalgebäudes könnte man zum Beispiel eine Belegung definieren, die nur das Personal enthält, sowie eine Belegung, bei der in den Hörsälen eine hohe Personendichte herrscht. Eine Belegung wird durch ein Objekt der Klasse `Assignment` repräsentiert. Das Projekt kann beliebig viele Belegungen enthalten, es kann aber maximal eine als aktuell ausgewählt werden. Es sind dann automatisch nur die `AssignmentAreas` im Plan enthalten, die zur aktuell gewählten Belegung gehören.

In einer Belegung werden sowohl die Orte, an denen sich Menschen aufhalten können, als auch die Eigenschaften der Personen gespeichert. Für die Angabe der Positionen der Personen werden Belegungsbereiche verwendet (Klasse `AssignmentArea`, siehe Abschnitt 5.1.2). Jedes `Assignment` enthält ein oder mehrere `AssignmentAreas`. Zu jedem Belegungsbereich kann man die Anzahl der Personen in diesem Bereich angeben, sowie den Typ der Personen. Sämtliche Personen in einem Bereich sind dann vom gleichen Typ.

Der Typ einer Personengruppe ist immer ein Objekt der Klasse `AssignmentType`. Jedes dieser Objekte repräsentiert einen Typ von Menschen mit gewissen Eigenschaften, die über Wahrscheinlichkeitsverteilungen angegeben werden können. Die tatsächlichen Personen werden beim Ausführen von Algorithmen zufällig nach den gewünschten Parametern erzeugt. Dazu muss (nachdem der Plan gerastert worden ist) eine **konkrete Belegung** (Klasse `ConcreteAssignment`) erzeugt werden. Durch das Erzeugen einer konkreten Belegung werden auch die tatsächlichen Positionen der Personen festgelegt (die Belegungsbereiche geben nur den Umkreis an, in dem sich die Personen befinden). Die konkrete Belegung kann von den Graphalgorithmen und dem zellulären Automaten verwendet werden.

Falls Personen einzeln positioniert werden sollen, muss für jede Person ein entsprechend kleines Gebiet plaziert werden. Auch die Eigenschaften der Personen können deterministisch gewählt werden, wenn das gewünscht ist. Da üblicherweise keine genauen Aussagen über den Aufenthaltort und die genauen Eigenschaften von Personen gemacht werden können, ist ein zufälliges Verfahren (außer zu Testzwecken) sinnvoller.

### 5.1.4. Personenanzahlen

Die Anzahl der Personen, die sich in einer `AssignmentArea` befindet, wird als ganzzahliger Wert mit der Methode `setEvacuees( int )` gesetzt. Für Belegungstypen kann eine Standardanzahl an Personen festgelegt werden. In diesem Fall wird jedem erzeugten Bereich diesen Typs diese Anzahl an Personen zugeordnet, sie kann jedoch später noch abgeändert werden. Die Anzahl der Personen in einem Bereich ist beschränkt durch die Größe des Bereiches zusammen mit einem Standardwert für die Fläche, die eine einzelne Person einnimmt.

Bevor eine konkrete Belegung erzeugt wird, wird geprüft, ob die festgelegte Personenzahl zu groß ist und gegebenenfalls eine Exception ausgelöst.

### 5.1.5. Zusammenhang von Belegungsbereichen und -typen

Der zu einem Belegungsbereich gehörende Typ kann mit der Methode `setAssignmentType( AssignmentType )` festgelegt werden. Ein Bereich trägt sich automatisch in eine Liste sämtlicher Bereiche eines Typs im zugehörigen `AssignmentType` ein. Falls der Bereich gelöscht wird, trägt er sich wieder aus. Andersherum führt das Löschen des Belegungstyps (oder gar der ganzen Belegung) dazu, dass auch zugehörige Belegungsbereiche gelöscht werden.

#### Unterstützte Eigenschaften

Jeder Belegungstyp definiert bestimmte Eigenschaften, die beim Erzeugen einer konkreten Belegung an die tatsächlich erzeugten Personen weitergegeben werden. Diese Eigenschaften werden durch Zufallsverteilungen festgelegt. Dabei gibt man u.a. die gewünschte Varianz an und kann eine deterministische Verteilung erhalten, indem man als Varianz Null wählt.

Die Verteilungen der Werte der Eigenschaften für einen bestimmten Personentyp werden mit Hilfe der Klasse `DistributedParameter` beschrieben. Diese Klasse stellt das Verhalten einer Zufallsvariable zur Verfügung. Jede Verteilung muss auf jeden Fall einen minimalen und maximalen Wert, der angenommen werden kann, erhalten. Das verhindert, dass fehlerhafte Werte entstehen, wie z.B. ein negatives Alter, zu hohe Geschwindigkeiten o.ä. Die Ergebnisse des Zufallszahlengenerators außerhalb des zulässigen Intervalls werden verworfen. Diese Lösung haben wir gewählt, da man z.B. die Normalverteilung nicht passend skalieren kann und durch Wahl von Randwerten als Ersatz für Werte außerhalb des Intervalls eine unzulässige Häufung auftreten würde.

Die Klasse `NormallyDistributedParameter` repräsentiert eine Normalverteilung, die durch die Parameter **Varianz** und **Erwartungswert** eindeutig festgelegt wird. Die Werte können mit den Methoden `setVariance( float )` und `setExpectedValue( float )` gesetzt werden. Dabei ist zu beachten, dass die

Varianz nicht negativ sein darf. Die Werte werden um den Erwartungswert herum liegen, die Breite der Streuung hängt von der Varianz ab. Falls die Varianz Null ist, wird immer der Erwartungswert erzeugt.

Eine Gleichverteilung auf einem Intervall wird durch die Klasse `UniformlyDistributedParameter` dargestellt. Sie ist bereits durch Angabe von minimalem und maximalem Wert eindeutig bestimmt.

Die Personen haben folgende Eigenschaften:

**Alter** Hierbei handelt es sich um die Altersverteilung der zu evakuierenden Personen. Es macht offensichtlich einen Unterschied, ob ein Krankenhaus oder ein Kindergarten evakuiert wird. Das Alter kann mit `setAge( DistributedParameter )` gesetzt werden. Vom Alter abhängig können andere Eigenschaften der Personen berechnet werden. Diese Berechnung sollte zufällig erfolgen, so dass Personen gleichen Alters z.B. auch verschiedene Geschwindigkeiten und Fitnesswerte erhalten können.

**Panik** Dies ist die Wahrscheinlichkeit, mit der eine Person in eine Panik verfällt. Dabei wird angenommen, dass panische Personen sich nicht angemessen verhalten. Sie können völlig ziellos handeln oder aber auch in Lethargie verfallen.

**Ortskenntnis** Diese Eigenschaft beschreibt, wie gut sich die entsprechende Person im zu evakuierenden Gebäude auskennt. Mit der Methode `setFamiliarity()` kann ein Wert für die Ortskenntnis festgelegt werden. Je besser die Ortskenntnis ist, desto besser ist der von der Person gewählte Fluchtweg. Dieser gewählte Fluchtweg ist dann allerdings nur aus Sicht der Einzelperson besser, global kann er durchaus noch sehr ungünstig sein, nämlich z.B. wenn ihn sehr viele andere Personen ebenfalls nehmen und somit Staus entstehen. Auch hier ist es sinnvoll, Wahrscheinlichkeiten zu wählen, um weiter gestreute Ergebnisse zu erhalten.

**Entscheidungsfreudigkeit** Entscheidungsfreudige Personen entscheiden sich für einen Weg und gehen ihn anschließend, während weniger entscheidungsfreudige Personen zögerlich agieren und oft ihre Routen ändern. Dieser Wert kann mit der `setDecisiveness( DistributedParameter )`-Methode festgelegt werden.

**Durchmesser** Der Durchmesser modelliert die Schulterbreite. Dies ist typischerweise ein Wert zwischen 40 und 70 cm. Der Wert wird im Allgemeinen nur für stetige Berechnungsmodelle benötigt, der zelluläre Automat verwendet eine Schulterbreite von 40 cm. Da wir stetige Berechnungsmodelle nicht mehr implementiert haben, ist dieser Wert bei uns auf 40 cm festgelegt.

Der wichtigste Faktor für eine Evakuierung ist die Geschwindigkeit der Personen. Ein guter Wert kann mit Experimenten gemessen oder einschlägiger Literatur entnommen werden. Die Geschwindigkeit korreliert üblicherweise stark mit dem Alter. Es muss aufgepasst werden, dass kein zu hoher Wert gewählt wird. Auch wenn Menschen versuchen, während einer Evakuierung zu rennen, ist die tatsächlich erreichte Geschwindigkeit oft bedeutend geringer.

### 5.1.6. Erzeugbare Datenstrukturen

Aus den beschriebenen Datenstrukturen können die Datenstrukturen **Z-Graph** und **Z-Zellulärer Automat** erzeugt werden. Damit gehören diese Datenstrukturen zwar zum Z-Format, dienen aber nicht wie die Plan- und Belegungsstrukturen als *Zwischen*format.

**Z-Graph** Die Datenstruktur für Graphen wird aus dem Z-Plan erzeugt. Anschließend wird (mit dem Umweg über eine konkrete Belegung) die Z-Belegung hinzugefügt. Die Graphdatenstrukturen werden in Abschnitt 5.3 beschrieben.

**Z-Zellulärer Automat** Auch die Datenstruktur für zelluläre Automaten wird aus dem Z-Plan erzeugt. Anschließend wird aus einer Z-Belegung eine konkrete Belegung gewonnen und die Aufenthaltsorte der Personen im Automaten gespeichert. Mehr über zelluläre Automaten findet sich in Abschnitt 5.2.

## 5.2. Der Zelluläre Automat

### 5.2.1. Allgemeine Zelluläre Automaten

Ein Zellulärer Automat ist ein Modell zur Durchführung einer zeitdiskreten Simulation. Das Modell basiert auf der Unterteilung eines Gebietes in benachbarte Zellen, welche jeweils einen Zustand aus einer endlichen Zustandsmenge besitzen. Der Übergang zwischen den Zuständen geschieht gemäß einer Übergangsfunktion.

### 5.2.2. Unser Zellulärer Automat

Unser Zellulärer Automat basiert auf Bauplänen von Gebäuden. Wir möchten diese Pläne in Zellen zerlegen, um die Evakuierung eines Gebäudes mit einem Zellulären Automaten simulieren zu können. Für die Unterteilung gibt es zwei grundlegende Verfahren: Zum einen kann der gesamte Bauplan als ein großes Gebiet aufgefasst und zerteilt werden, indem er mit einem rechteckigen Raster aus Zellen überdeckt wird. Aufgrund des rechteckigen Rasters werden dabei unter Umständen auch Teile des Plans überdeckt, die für die Simulation uninteressant sind, weil sie außerhalb des Gebäudes liegen. Eine solche Unterteilung ist wenig strukturiert und verschwendet Speicherplatz für Zellen, die für die Simulation irrelevant sind. Wir versuchen hingegen, die Gebäudestruktur in die Unterteilung einfließen zu lassen und soweit möglich nur die Innenbereiche des Gebäudes zu modellieren. Dazu unterteilen wir die Innenbereiche in einzelne Räume und verbinden diese mit Türen (ähnlich wie im Z-Format).

### 5.2.3. Klassenstruktur

Ein Bauplan wird von uns in Räume unterteilt (Klasse `Room`), welche sich wiederum aus Zellen zusammensetzen (Klasse `Cell`). Dabei gibt es verschiedene Arten von Zellen, welche sich hinsichtlich ihrer Funktion unterscheiden:

**Raumzellen** Raumzellen (Klasse `RoomCell`) modellieren Zellen ohne spezielle Funktion. Jede Zelle besitzt einen Geschwindigkeitsfaktor `speedFactor` im Intervall $[0, 1]$, der für normale Raumzellen mit dem Standardwert

1.0 belegt wird, d.h. Raumzellen können mit normaler Geschwindigkeit überquert werden.

**Realisierung von Verzögerungsbereichen** Für Verzögerungsbereiche gibt es keine eigene Zellenart, da man durch den `speedFactor` die gewünschte Verzögerung einstellen kann, d.h. Zellen in Verzögerungsbereichen werden mit Raumzellen modelliert, für die der Geschwindigkeitsfaktor angepasst wird. Kleinere Geschwindigkeitsfaktoren senken die Geschwindigkeit ab (ähnlich wie im Z-Format).

**Treppenzellen** Treppenzellen (Klasse `StairCell`) modellieren eine Treppe und besitzen zwei Geschwindigkeitsfaktoren, einen für Treppab und einen für Treppauf.

**Türzellen** Türzellen (Klasse `DoorCell`) realisieren die Verbindung zweier Räume: Über sie werden Räume betreten und verlassen. Eine Türzelle besitzte eine Liste von benachbarten Türzellen, die von dieser aus erreichbar sind (die ArrayList `nextDoors`). Ein Individuum kann eine Tür durchqueren, indem es eine Türzelle in einem Raum betritt und sich dann dafür entscheidet, zu einer Partnertürzelle zu wechseln. Für das Individuum ist die Partnertürzelle ein Nachbar der Türzelle, der sich von den neben der Türzelle im gleichen Raum liegenden Zellen nicht unterscheidet.

**Sichere Zellen** Auf sicheren Zellen (Klasse `SaveCell`) sind die Individuen in Sicherheit. Von sicheren Zellen bewegen sich Individuen nur noch zu anderen sicheren Zellen oder zu Ausgangszellen.

**Ausgangszellen** Über Ausgänge (Klasse `ExitCell`) verlassen Individuen das Gebäude und bringen sich in Sicherheit.

Jede Zelle ist entweder von einem Individuum (Klasse `Individual`) besetzt oder frei. Weiterhin kann der Weg zu jeder Nachbarzelle einer Zelle als „blockiert“ markiert werden: Individuen können dann die Zelle in dieser Richtung nicht verlassen. Diese Möglichkeit kann genutzt werden, um Möbel oder ähnliche Barrieren zu modellieren.

Um eine grobe Bewegungsrichtung der Individuen vorgeben zu können, verwenden wir **Potentiale** (Klasse `PotentialManager`). Ein Potential weist jeder

Zelle eine positive ganze Zahl zu, welche als Orientierungshilfe für die Individuen dient (sie können z.B. versuchen, auf Zellen mit möglichst kleinem Potentialwert zu gelangen). Jedes Individuum orientiert sich an einem bestimmten Potential, es kann jedoch mehrere Potentiale geben. Jedes Potential setzt sich aus lokalen Potentialen für jeden Raum zusammen (Klasse `LocalPotential`).

Wir unterscheiden statische Potentiale (Klasse `StaticPotential`) und dynamische Potentiale (Klasse `DynamicPotential`). Statische Potentiale werden zu Beginn der Simulation festgelegt und ändern sich anschließend nicht mehr. Dynamische Potentiale können sich während der Simulation ändern und können z.B. Individuen den Weg zu anderen Individuen(-gruppen) weisen. Mehr dazu in Abschnitt 7.1.4.

Alle Klassen werden von der Klasse `CellularAutomaton` verwaltet. Die Klasse stellt gleichzeitig die Schnittstelle für den Zugriff von außen dar.

Die Klasse `CellularAutomaton` und die von ihr verwalteten Klassen stellen nur die statische Struktur des zellulären Automaten dar. Das dynamische Verhalten wird von Simulationsalgorithmen hinzugefügt, die in Abschnitt 7.1.2 beschrieben werden.

### 5.2.4. Eigenschaftensätze

Man kann den benutzten Automaten mit Eigenschaftsdateien weiter anpassen, so kann man z.B. erwirken, dass die Individuen ihr Potential nicht ändern dürfen oder das sie sich besonders panisch verhalten.

## 5.3. Die Graph-Datenstruktur

Im Folgenden werden Datenstrukturen für Graphen beschrieben. Wir haben einerseits allgemeine Klassen und Interfaces definiert, um Standardstrukturen darzustellen und andererseits spezielle Klassen implementiert, die wir für die Anwendung von Flussalgorithmen benötigen.

### 5.3.1. Knoten und Kanten

**Knoten** und **Kanten** sind die Grundelemente von Graphen. Bei ihrer Implementierung muss besonders auf Effizienz geachtet werden, weil während der Ausführung von Flussalgorithmen oft auf einzelne Kanten und Knoten zugegriffen wird. Daher werden (an LEDA[3] orientiert) Knoten- und Kanten-IDs vergeben, so dass man Kanten- und Knotenmengen anhand der IDs effizient indizieren kann. Zur Verallgemeinerung dieses Konzepts wird das Interface `Identifiable` benutzt: Ein Objekt einer Klasse, die `Identifiable` implementiert, ist anhand einer ID identifizierbar, die man mit id() abfragen kann. Wir nutzen das folgendermaßen aus: Haben wir Objekte mit fortlaufenden IDs gegeben, so können diese in einem Array anhand der IDs gespeichert werden, ohne dass Hashing oder ähnliches notwendig ist. Wir organisieren das Erzeugen der Objekte so, dass dies immer der Fall ist.

Die Klassen `Node` und `Edge` implementieren `Identifiable`. Ein Objekt der Klasse `Edge` kann den zugehörigen Start- oder Endknoten zurückliefern sowie den einem Knoten bzgl. der Kante gegenüberliegenden Knoten.

### 5.3.2. Pfade

Unsere Datenstruktur kennt zwei verschiedene Arten von Pfaden: **Statische Pfade** (`StaticPath`) und **Dynamische Pfade** (`DynamicPath`). Beide erben von dem Interface `Path`, das einen Pfad beschreibt. Pfade enthalten Kanten und können nach vorne oder hinten um eine Kante verlängert oder verkürzt werden. Außerdem kann über Pfade bzw. ihre Kanten iteriert werden.

Ein statischer Pfad ist eine Abfolge von Kanten und somit eine relativ direkte Realisierung des Interfaces. Ein dynamischer Pfad bezieht außerdem noch ein, dass Fluss zwischen dem Passieren von Kanten in Knoten warten kann. Daher stellt `DynamicPath` neben den obigen Methoden noch Methoden für die Verwaltung der Wartezeiten zur Verfügung.

### 5.3.3. Graphen und Netzwerke

Ein **Graph** besteht aus einer Knoten- sowie einer Kantenmenge, die **gerichtet** oder **ungerichtet** aufgefasst werden kann. Zur Speicherung der beiden Mengen kann man entweder Arrays oder Listen verwenden, je nachdem, ob sich der Graph während der geplanten Anwendung häufig ändert oder eher statisch ist.

Das Interface `Graph` bietet den Ausgangspunkt für Graphen. Dieses schreibt Methoden vor, mit denen man die Eigenschaften eines konkreten Graphen abfragen kann.
Außerdem bietet das Interface nützliche Methoden für die Durchführung von Graphalgorithmen, die vor allem die Nachbarn von Knoten und Kanten sowie den Grad von Knoten betreffen. Außerdem kann der Graph zurückliefern, ob es einen Pfad zwischen zwei Knoten gibt.
Die Adjazenz- und Inzidenzlisten werden zur Beschleunigung intern zwischengespeichert.

Flussprobleme werden üblicherweise auf Netzwerken definiert. Wir verstehen unter einem **Netzwerk** einen gerichteten Graphen. Die Struktur des Eingabenetzwerkes ändert sich während der Ausführung eines Flussalgorithmus nicht oder nur wenig. Die Klasse `Network` implementiert daher einen gerichteten Graphen mit Hilfe einer arraybasierten Darstellung. Die Größe der Arrays sollte zu Beginn der Ausführung gesetzt und nicht wieder verändert werden. Falls doch Veränderungen auftreten, wird die Arraygröße angepasst.
Die Aussage, Graphen seien in Flussproblemen statisch, ist ein wenig geschummelt. Zwar werden in der Regel keine Knoten oder Kanten neu hinzugefügt, es kommt aber vor, dass sie von Algorithmen – vorübergehend – gelöscht werden. Um diesen Fall zu modellieren, ohne die statische Struktur des Netzwerks aufzugeben, unterstützt unser Netzwerk das **Verstecken** von Kanten. Versteckte Kanten und Knoten werden von den Ausgabemethoden des Netzwerkes (z.B. für Adjazenzlisten) ignoriert.

Unsere Datenstrukturen sehen noch zwei weitere spezielle Netzwerke vor, die von `Network` erben: **Residualnetzwerke** (`ResidualNetwork`) und **zeitexpandierte Netzwerke** (`TimeExpandedNetwork`).

**Residualnetzwerke** beziehen sich immer auf einen Fluss im Netzwerk. Ist

der zugehörige Fluss der Nullfluss, so entspricht das Residualnetzwerk dem eigentlichen Netzwerk. Ansonsten besteht das Residualnetzwerk aus Kanten des Originalnetzwerkes und Rückwärtskanten: Fließt auf einer Kante Fluss, so enthält das Residualnetzwerk die entgegengesetzte Kante, deren Kapazität der Flussmenge entspricht. Kanten, die im ursprünglichen Netzwerk vorhanden sind, gibt es auch im Residualnetzwerk, wenn ihre Kapazität durch den Fluss noch nicht aufgebraucht ist. Ihre Kapazität ist dann die verbleibende Kapazität. Die Fahrzeiten von Kanten im Residualnetzwerk entsprechen bei ursprünglichen Kanten deren normalen Fahrzeiten, bei Rückwärtskanten wird die Fahrzeit mit (-1) multipliziert. Die Klasse `ResidualNetwork` unterstützt das Augmentieren des aktuellen Flusses auf einer Kante um einen Flusswert und bietet Abfragemöglichkeiten für die Kapazitäten, Fahrzeiten und den aktuellen Fluss im Netzwerk.

**Zeitexpandierte Netzwerke** fertigen für jeden Zeitschritt bis zu einem gegebenen Zeithorizont eine Kopie des eigentlichen Graphen an und verbinden die Knoten der verschiedenen Schichten entsprechend der Fahrzeiten auf den Kanten. Viele dynamische Flussprobleme lassen sich auf statische Flussprobleme im zeitexpandierten Netzwerk zurückführen. Ein zeitexpandiertes Netzwerk gehört zu einem ursprünglichen Graphen und besitzt neben den Kopien der ursprünglichen Quelle(n) und Senke(n) eine eigene Quelle und Senke. Diese Informationen können in der Klasse `TimeExpandedNetwork` ausgelesen werden. Außerdem macht es für das zeitexpandierte Netzwerk einen Unterschied, ob Fluss in Knoten des ursprünglichen Netzwerks warten darf: In diesem Fall kann man von einem Knoten immer mit beliebiger Rate zur entsprechenden Kopie der nächsten Zeitscheibe gelangen (d.h. das Warten wird durch Pfade simuliert, die in mehreren aufeinanderfolgenden Zeitschichten durch Kopien des gleichen Knotens gehen). Unsere Klasse stellt hierfür eine Option zur Verfügung. Außerdem bietet sie die Möglichkeit, einen (statischen) Pfad im zeitexpandierten Netzwerk in einen dynamischen Pfad im ursprünglichen Netzwerk umzurechnen.

### 5.3.4. Funktionen

Zuordnungen spielen für Flussalgorithmen an verschiedenen Stellen eine Rolle. Wir benennen die Klassen für verschiedene Funktionstypen entsprechend ih-

res Definitions- und Wertebereichs: Eine Klasse für Funktionen, die Bytes auf Strings abbilden würden, hieße hier ByteStringMapping.

Bei der Modellierung der Eingabedaten möchten wir in Graphen Werte auf Kanten und/oder Knoten speichern, d.h. wir möchten z.B. eine Funktion definieren, die jeder Kante eine Kapazität zuordnet. Andererseits variiert je nach Algorithmus, welche Werte tatsächlich gespeichert werden sollen (z.B. Probleme mit oder ohne Angeboten/Bedarfen auf den Knoten). Daher werden die Kapazitäten, Fahrzeiten und ähnliches nicht innerhalb der Graphenklasse als Attribute gespeichert. Stattdessen wird die Funktionenklasse `IdentifiableIntegerMapping` verwendet: Einem `Identifiable`-Objekt (also einer Kante oder einem Knoten) wird ein `Integer` zugewiesen.

Auch bei der Speicherung von Flüssen werden Funktionen benötigt. Möchte man z.B. einen `FlowOverTime` (siehe Abschnitt 5.3.6) speichern, indem man für jede Kante eine Zuordnung von Zeitpunkten auf Flusswerte angibt, so benötigt man zwei Funktionen: Für eine bestimmte Kante braucht man ein `IntegerIntegerMapping`, um Zeitpunkte auf Flusswerte abzubilden (dabei werden natürlich nur die Zeitpunkte verwaltet, zu denen sich der Flusswert ändert). Um den Kanten eine solche Funktion zuzuweisen, verwenden wir ein `IdentifiableObjectMapping`, mit dem man einer Kante ein beliebiges Objekt (also auch ein `IntegerIntegerMapping`) zuweisen kann. `IdentifiableObjectMappings` können auch für andere Zwecke verwendet werden, z.B. wenn man für jeden Knoten in einem Graphen einen Vorgänger speichern möchte – man kann mit einem solchen Mapping also beliebige Dinge auf Kanten und Knoten speichern.

Die Speicherung der Funktionen erfolgt arraybasiert. Bei den Mappings mit Definitionsbereich `Integer` können die `Integer` direkt als Indizierung verwendet werden, bei Definitionsbereich `Identifiable` können die IDs verwendet werden (hier nutzen wir fortlaufende Knoten- und Kanten-IDs aus). Durch die arraybasierte Speicherung sind die Klassen schnell.

Für `IntegerIntegerMappings` stellen wir zusätzliche Funktionalitäten bereit: Diese können addiert und subtrahiert werden, außerdem kann man über ein solches Mapping integrieren und erhält wiederum ein `IntegerIntegerMapping`.

### 5.3.5. Weitere Klassen

Wie oben bereits erwähnt, gibt es prinzipiell die Möglichkeit, `Identifiable`-Objekte **arraybasiert** oder **listenbasiert** zu speichern. Für beide Möglichkeiten sehen wir eine Datenstruktur vor.

Um die Spezifikation anderer Klassen davon unabhängig zu halten, wird das Interface `IdentifiableCollection` benutzt. Dieses legt die Standardfunktionalitäten fest.

Das `ArraySet` basiert auf einem Array. Es implementiert eine Menge, d.h. Elemente können nur einmal enthalten sein. Da `Identifiable`-Objekte dann gleich sind, wenn sie die gleiche ID haben, bedeutet dies, dass das `ArraySet` nur ein Element pro ID enthalten kann. So können die Elemente anhand ihrer IDs indiziert werden. Die `ListSequence` basiert auf einer `LinkedList` und unterstützt vor allem das Speichern einer Reihenfolge. Dies ist z.B. für Pfade wichtig, deren Kanten man nicht in einem `ArraySet` speichern könnte: Im `ArraySet` entspricht die Reihenfolge der Elemente immer der ihrer IDs. Außerdem kann eine `ListSequence` Elemente doppelt enthalten.

Eine weitere Implementierung der `IdentifiableCollection` ist das `HidingSet`. Das `HidingSet` ist von `ArraySet` abgeleitet und bietet die Funktionalität, die wir zum Verstecken von Elementen benötigen: In einem weiteren Array wird gespeichert, welche Elemente gerade versteckt sind. Ein `HidingSet` verhält sich bezüglich der Standardfunktionalitäten, als wären die versteckten Elemente nicht vorhanden. Abgesehen davon lässt es sich in eine `IdentifiableCollection` verwandeln, die die versteckten Elemente nicht enthält.

Schließlich benötigen wir noch eine vierte Implementierung der `IdentifiableCollection`. Der Grund ist der Folgende: Eine in einem Netzwerk zwischengespeicherte Inzidenzliste soll bei der Benutzung Elemente ignorieren, die im ebenfalls im Netzwerk gespeicherten `HidingSet` zur Verwaltung der Kanten gerade versteckt sind. Die Kanten in jeder einzelnen Inzidenzliste einzeln zu verstecken, bereitet unnötigen Aufwand und birgt die Gefahr der Inkonsistenz. Stattdessen benutzen wir eine `DependingListSequence`: Diese erweitert die `ListSequence` um eine Abhängigkeit von einem `baseSet`. Die Ausgabemethoden der `DependingListSequence` ignorieren Elemente, die zwar in der Liste vorhanden sind, aber nicht im `baseSet`. Verwendet man ein `HidingSet` als

`baseSet`, so sind versteckte Elemente „nicht vorhanden“ und werden von der `DependingListSequence` ignoriert.

### 5.3.6. Dynamische Flüsse

Für die Ergebnisspeicherung bei dynamischen Flussalgorithmen benötigen wir Datenstrukturen für **dynamische Flüsse**. Diese Datenstrukturen werden dann im **BatchResult** benutzt und zusammen mit dem Rest der Ergebnisse in den .ers-Dateien abgespeichert.

Ein dynamischer Fluss ordnet jeder Kante zu jedem Zeitpunkt bis zum gegebenen Zeithorizont einen Flusswert zu. Eine **kantenbasierte** Flussspeicherung setzt dies direkt um und speichert für jede Kante eine Zuordnung von Zeitpunkten auf Flusswerte. Dafür wird ein `IntegerIntegerMapping` benötigt (siehe Abschnitt 5.3.4). Unsere Klasse für kantenbasierte Flussspeicherung heißt `FlowOverTime`.

Alternativ kann man Flüsse auch **pfadbasiert** speichern. Aufgrund der Flusserhaltung kann man den Gesamtfluss (sofern es nur ganzzahlige Flusswerte gibt) immer in eine endliche Menge von Pfaden von Quellen zu Senken aufteilen, so dass auf einem Pfad Fluss mit einer festen Rate fließt. Die Klasse `PathFlow` stellt einen Pfad dar, auf dem Fluss fließt. Intern wird ein `DynamicPath` benutzt (siehe Abschnitt 5.3.2). Um darzustellen, dass der Fluss auf einem Pfad nicht zum Zeitpunkt Null zu fließen beginnt, kann man innerhalb des `DynamicPath` eine Wartezeit vor der ersten Kante spezifizieren. Außerdem speichert ein PathFlow, wie groß die Flussmenge ist, die insgesamt geschickt werden soll. Zusammen mit der Rate ergibt sich daraus der Zeitpunkt, zu dem kein Fluss mehr geschickt wird. Die Klasse `DynamicFlow` kapselt alle zu einem dynamischen Fluss gehörenden `PathFlows`.

## 5.4. Batchverarbeitung und Ergebnisspeicherung

Zum Ausführen mehrerer Simulations- oder Optimierungsdurchläufe wurde eine Batchverarbeitung implementiert. Es ist damit ebenfalls möglich, mehrere Belegungen in einem Gebäude zu simulieren und auch mehrere Projekte nacheinander auszuführen. Dazu werden einige Klassen benötigt, um die Batches

zu verwalten. Die Grundsätzliche Aufteilung ist zwischen Eingabe- und Ausgabeobjekten, die im wesentlichen eine Liste der Jobs repräsentieren.

Zu jedem Eingabeobjekt wird, wenn der Batch ausgeführt wird, ein entsprechendes Ausgabeobjekt erzeugt. Die Klasse `Batch` sammelt mehrere Aufgaben vom Typ `BatchEntry`. Sie fungiert als Container und erlaubt es alle Aufgaben gleichzeitig auszuführen. Die Aufgaben werden dabei nacheinander ausgeführt.

Ein `BatchEntry` kapselt alle Informationen, die nötig sind, um die Simulation und Optimierung eines Projektes durchzuführen. Dazu gehört die ausgewählte Belegung, die Anzahl der Durchläufe und die Angabe, ob überhaupt eine Simulation oder Optimierung durchgeführt werden soll. Ein Eintrag kann mit der Methode `execute()` gestartet werden.

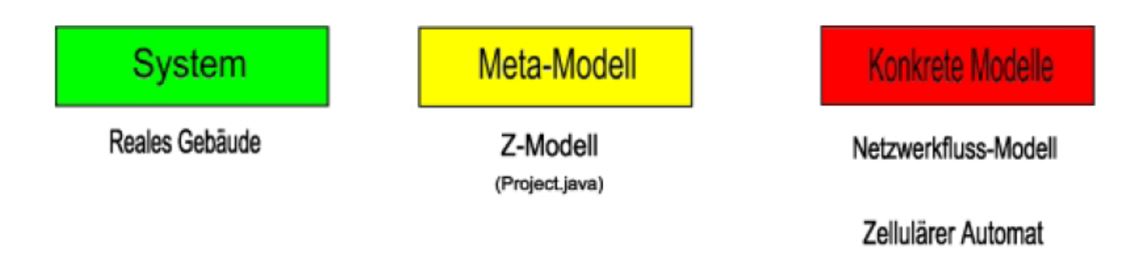
Die Ergebnisse der Algorithmen werden in der Klasse `BatchResult` gespeichert. Sie ist ein Analogon zu `Batch` und stellt eine Sammlung von `BatchResultEntry`-Objekten dar. Ein `BatchResultEntry` kapselt die Ausgaben zu einem Eintrag in der Batchliste. Das können je nach gewählten Eigenschaften ein dynamischer Fluss und / oder Informationen über einen Simulationslauf mit dem Zellulären Automaten sein.

Die berechneten Informationen können abschließend in einer Ergebnisdatei gespeichert werden, die noch zusätzliche Informationen zur Visualisierung enthält. Wenn diese Datei später wieder geladen wird, können die Ergebnisse in der Statistik oder der Visualisierung angesehen werden.

# 6. Umwandlung des Z-Formats

## 6.1. Struktur der Umwandlungen

Für die Umwandlung des Z-Formats in andere Datenstrukturen haben wir uns eine allgemeine Struktur überlegt. Die folgende Graphik liefert zunächst einen Überblick über die verschiedenen Abstraktionsebenen in unserem Evakuierungssystem:




Das zu beschreibende reale System ist das zu evakuierende Gebäude, das als Bild vorgegeben ist. Dieses System muss in ein Modell übertragen werden, mit dem das Programm umgehen kann. Das Modell dient in unserem Fall auch als Zwischenformat, da später mehrere Anwendungen verschiedene weitere Abstraktionsebenen voraussetzen, nämlich Graphalgorithmen und Zelluläre Automaten.

Für diese Anwendungen muss das Z-Format, welches ein solches Modell ist, zur weiteren Verarbeitung in weitere Modelle umgeformt werden. Eben dazu dienen die hier beschriebenen Konverter. Im Wesentlichen beschreiben sie Abbildungen von Z-`Project`-Dateien in Zelluläre Automaten bzw. Graphen und berechnen notwendige Zusatzinformationen.

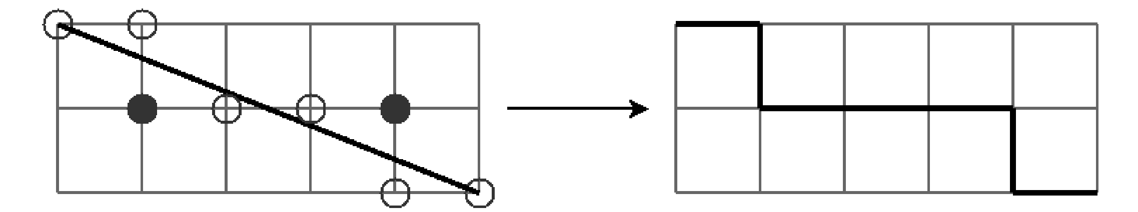


**Abbildung 6.1.:** Rasterung einer schrägen Kante.

## 6.2. Rasterung des Z-Formats

Im Z-Format werden alle Räume zugelassen, die sich als schnitt- und lochfreies Polygon modellieren lassen. Der Zelluläre Automat benötigt aber eine Gitterdarstellung der Räume, die aus Konsistenzgründen auch bei der Umwandlung in Graphen verwendet werden soll. Die Rasterung muss somit Räume auf eine Gitterstruktur abbilden.

Wir gehen dazu zweistufig vor. Zuerst werden alle Elemente des Z-Formats so verändert, dass sie nur noch auf Gitterpunkten und -linien liegen. Die Datenstruktur bleibt dabei zunächst gleich, aber die Polygone selbst verändern sich. Erst im Anschluss wird dann eine Gitterdarstellung erzeugt. Jetzt verändert sich der dargestellte Plan nicht mehr, die Informationen werden nur in eine anderen Datenstruktur abgebildet.

### 6.2.1. Rastern

Wir möchten also alle Elemente des Z-Plans so verändern, dass sie nur noch auf einem gegebenen Raster verlaufen. Diese Aufgabe ist leider hochgradig nicht-trivial und bei der Umsetzung treten eine Reihe von Schwierigkeiten bzw. Ausnahmen und Sonderfällen auf.

Zu Beginn verschieben wir die Anfangs- und Endpunkte aller Kanten im Z-Format auf die jeweils nächsten Gitterpunkte. Im Anschluss müssen einige Sonderfälle behandelt werden:

**Schräge Wände** Selbst wenn die Eckpunkte aller Kanten auf dem Raster liegen, müssen die Kanten selbst nachbearbeitet werden, sofern sie nicht zufällig auf dem Raster verlaufen.

Diese Aufgabe ist im Prinzip noch relativ einfach zu lösen. Eine schräge Kante im Z-Format wird mittels Matrixtransformation in eine standardisierte Lage gebracht. Anschließend wird ein Kantenzug errechnet, durch den die Kante ersetzt wird. Dies geschieht durch Identifikation der Gitterpunkte, die die Kante am Besten approximieren (siehe Abbildung 6.1). Dazu wird das Gitter in x-Richtung durchlaufen und für jede x-Gitter-Achse wird ein y-Wert gewählt, der dem Kantenverlauf am besten entspricht (nicht ausgefüllte Kreise). Alle Punkte, die auf der gleichen y-Ebene liegen, können leicht horizontal verbunden werden. So entstehen kleine, zur x-Achse parallele, "Mini-Kanten", die jedoch noch ebenenweise verbunden werden müssen, um einen zusammenhängenden Kantenzug zu erhalten. Dies geschieht wiederum so, dass die ursprüngliche Kante so gut wie möglich approximiert wird (ausgefüllte Kreise). Durch eine weitere Matrixtransformation wird der neue Kantenzug wieder in die ursprüngliche Lage gebracht.

**Nullräume** Es kann vorkommen, dass kleine Räume (zum Beispiel solche, die eine Tür darstellen), nach der Rasterung keine Fläche mehr besitzen. Solche Räume sind unsinnig und können je nach Behandlung innerhalb der Algorithmen auch Fehler auslösen. Daher werden sie nach der Rasterung entfernt.

**Andere Artefakte** Viele weitere Sonderfälle machen die Rasterung zu einem schwierigen Thema. Allein die Behandlung von nullflächigen Türen ist zu kompliziert, um automatisch gelöst zu werden. In solchen Fällen ist eine Nachbearbeitung durch den Nutzer notwendig.

### 6.2.2. Gitterdarstellung

Nach der erfolgreichen Rasterung der Elemente des Z-Formats wird eine Gitterdarstellung des gerasterten Plans erstellt.

Die dafür verwendete Datenstruktur besteht aus relativ vielen verschiedenen Klassen. Diese lassen sich in drei Gruppen einteilen:

- Klassen, die den gesamten Plan beschreiben. Die grundlegendste Klasse hier heißt `RasterContainer` und stellt einen Container für gerasterte

Räume dar. Die abgeleiteten Klassen `ZToCARasterContainer` und `ZToGraphRasterContainer` modellieren geringfügige Unterschiede, die sich für die weitere Umwandlung in Zelluläre Automaten und Graphen ergeben.

- Klassen, die einen gerasterten Raum beschreiben. Die Klasse `Raster` beschreibt zunächst ein Gitter. In einem `RoomRaster` kann man zusätzlich speichern, welche Teile des Gitters tatsächlich zum Raum gehören und welche Attribute die einzelnen Gitterquadrate haben (z.B. können sie Teil eines Verzögerungsbereichs sein). Spezielle unterschiedliche Attribute von Gitterquadraten werden durch die abgeleiteten Klassen `ZToCARoomRaster` und `ZToGraphRoomRaster` modelliert.
- Klassen, die einzelne Gitterquadrate beschreiben. Ein solches Quadrat wird durch die Klasse `RasterSquare` beschrieben. Die Klasse `RoomRasterSquare` erweitert das `RasterSquare` um Attribute (z.B. Teil eines Verzögerungsbereiches zu sein). Weiterhin existieren die abgeleiteten Klassen `ZToCARoomRasterSquare` und `ZToGraphRoomRasterSquare`.

Die Oberklassen sind immer generisch.

Um die Gitterdarstellung zu erzeugen, gehen wir die einzelnen Räume durch und stellen zunächst fest, wie groß die Gitter sein müssen und welche Gitterquadrate betretbar sind. Anschließend werden alle besonderen Bereiche durchgegangen, dabei werden die daraus hervorgehenden Attribute in den einzelnen Gitterquadraten eingetragen. Außerdem müssen Türen und Teleportkanten eingetragen werden (dies geschieht ebenfalls über Eigenschaften der einzelnen Quadrate).

Die so erzeugte Datenstruktur kann nun als Ausgangspunkt für die Umwandlung in den Zellulären Automaten und den Graphen verwendet werden.

### Umwandlung der Treppen

Treppen stellen bei der Umwandlung eine besondere Herausforderung dar, da es nicht ausreicht, ein Gitterquadrat als „Treppe“ zu markieren. Zusätzlich muss gespeichert werden, wie schnell Individuen sich auf dem Gitterquadrat bewegen können, wenn sie „hinauf“ oder „hinab“ gehen, und welche Übergänge zwischen Quadraten überhaupt „hinauf“ oder „hinab“ verlaufen. Dazu besitzt

jedes Quadrat eine `Map`, die jeder Richtung ein `Level` zuweist: `higher, equal` oder `lower`, entsprechend liegt das Rasterquadrat, was man in dieser Richtung erreichen kann, höher, gleichhoch oder tiefer als das eigene. Außerdem werden in jedem Quadrat zwei Geschwindigkeitsfaktoren für aufwärts und abwärts gehende Individuen verwaltet, so dass bei für einen Übergang zwischen zwei Rasterquadraten der passende Geschwindigkeitsfaktor ermittelt werden kann.

Im Z-Format sind Treppen Polygone mit zwei ausgezeichneten Kantenzügen, nämlich *unten* und *oben*. In der Umwandlung werden daraus für alle Gitterquadrate die Levelinformationen berechnet. Dazu wird ein Algorithmus benutzt, der ähnlich arbeitet wie der Algorithmus zur Bestimmung statischer Potentiale, der in Abschnitt 7.1.4 beschrieben wird.

## 6.3. Umwandlung für den Zellulären Automaten

Die in 6.2.2 beschriebene Datenstruktur ist bereits so aufgebaut, dass die Umwandlung in den Zellulären Automaten nur noch wenig Arbeit bereitet:

### 6.3.1. Umwandlung der Gitterdatenstruktur

#### Umwandlung der Raumgitter

Für jedes gerasterte Raumgitter vom Typ `ZToCARoomRaster` wird im Zellulären Automaten ein Raum der Klasse `Room` erstellt. Die in den Raumgittern enthaltenen Gitterquadrate des Typs `ZToCARasterSquare` können direkt in Zellen vom Typ `Cell` umgesetzt werden. Jedes Gitterquadrat eines Gitterraumes entspricht dabei genau einer Zelle im zellulären Automaten.

#### Umwandlung der Attribute der Gitterquadrate

Die Informationen aus den Attributen der Gitterquadrate werden ebenfalls in den Zellulären Automaten übernommen. Dazu gehören neben den Bereichsinformationen (Gitterquadrat ist Teil eines sicheren Bereichs, eines Verzöge-

rungsbereichs usw.) auch Kennzeichnungen von Türen, Treppen und Barrieren.

Die Bereichsinformationen werden im Zellulären Automaten durch eigene Unterklassen der Klasse `Cell` ausgedrückt, die Barrieren werden als Attribute der Zellen gespeichert.

Während *Türen* im Z-Format von Tür*kanten* dargestellt werden, werden sie im Zellulären Automaten als Tür*zellen* modelliert. Um hier die Umrechnung zu ermöglichen, werden im `ZToCARoomRaster` alle Gitterquadrate gekennzeichnet, die an eine Türkante (d.h. an eine passierbare Kante oder an eine Teleportkante) angrenzen.

Für *Treppen* gibt es im Zellulären Automaten `StairCells`, die jeweils zwei zusätzliche Geschwindigkeitsfaktoren `speedFactorUp` und `speedFactorDown` enthalten. Außerdem enthält jede `StairCell` eine `Map`, die jeder Richtung ein `Level (higher, equal, lower)` zuordnet. Verlässt man eine Zelle in eine bestimmte Richtung, so wird zuerst getestet, ob das Rasterquadrat in dieser Richtung höher, gleichhoch oder tiefer liegt. Entsprechend wird der Geschwindigkeitsfaktor ausgewählt. Die Informationen für Treppenzellen können direkt aus der Gitterquadratstruktur übernommen werden.

Die Umwandlung von *Barrieren* ist etwas komplizierter. Grundsätzlich werden Barrieren als Attribute der Zellen des Zellulären Automaten gespeichert: In jeder Zelle kann der Weg zu jeder der acht Nachbarzellen als blockiert markiert werden. Verläuft also zwischen zwei Zellen eine Barriere[1], so markieren wir in beiden Zellen den Weg zur jeweils anderen Zelle als blockiert, siehe dazu Abbildung 6.2. Dabei kann es insbesondere passieren, dass Übergänge zu den diagonalen Nachbarzellen blockiert werden.

Wir müssen also herausfinden, zwischen welchen Zellen eine Barriere verläuft. Dies können wir erreichen, indem wir zu jeder Barriere des Z-Formates diejenigen Zellen bestimmen, die sich oberhalb und unterhalb bzw. links und rechts der Barriere befinden und anschließend alle Wege zwischen ihnen blockieren. Stoßen wir auf Barrieren, die aneinander grenzen, behandeln wir sie, als wären sie eine einzige, zusammengehörige Barriere, d.h. wir erlauben nicht, dass sich Individuen zwischen zwei Barrieren hindurchzwängen.

[1]Da wir alle Elemente des Bauplans auf das Raster verschieben, können keine Barrieren durch die Zellen laufen

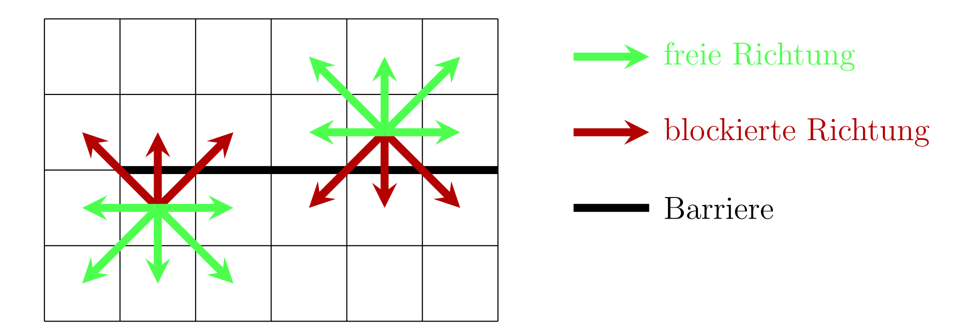



**Abbildung 6.2.:** Eine Barriere blockiert den Weg zwischen einigen Zellen. Die Individuen können weder durch die Barriere laufen, noch können sie schräg an ihr vorbeilaufen.

### 6.3.2. Umwandlung der Individuenbelegungen

Für die Umwandlung der Individuenbelegungen des Z-Formats in entsprechende Belegungen des Zellulärenautomaten müssen zum einen die Positionen der Personen und zum anderen ihre Eigenschaften umgerechnet werden.

#### Umwandlung der Individuenpositionen

Das Z-Format stellt konkrete Individuenbelegungen[2] bereits in einer Form zur Verfügung, in der sich in jedem Gitterquadrat nur ein Individuum befindet. Da jedes Gitterquadrat genau einer Zelle im Zellulären Automaten entspricht, genügt es nun, aus den Koordinaten jedes Individuums sein Gitterquadrat zu berechnen. Da bei der Umrechnung des Z-Plans in den Zellulären Automaten auch eine Zuordnung der Gitterquadrate auf Zellen gespeichert wird, kann nun das Individuum direkt auf die passende Zelle im Zellulären Automaten gesetzt werden.

### 6.3.3. Umwandlung der Individuenparameter

**Attribute im Z-Format** Das Z-Format sieht für jedes Individuum die Attribute **diameter, age, familiarity, panic** und **decisiveness** vor. Im Zellulären Automaten hingegen existieren für Individuen die Attribute **age, familarity, panic, panicFactor, slackness, exhaustion, exhaustionFactor, maxSpeed, currentSpeed** und **reactiontime**. Um diese Attribute verwenden zu

[2]Das sind Belegungen, in denen die Position der Individuen nicht *abstrakt* durch Verteilungen angegeben wird, sondern jedes Individuum bereits eine *konkrete* Position besitzt.

können, müssen wir zunächst die Attribute aus dem Z-Format auf die Attribute im Zellulären Automaten abbilden. Dabei brauchen wir uns um das Attribut **diameter** nicht zu kümmern, da die Personen im Zellulären Automaten eine festgelegte Größe haben. Einige Attribute werden direkt übernommen, während die Umrechnung von anderen Attributen durch das aktuell gewählte `ParameterSet` bestimmt wird (Näheres dazu siehe 7.1.7). Sofern nicht anders angegeben, haben wir die Umrechnungen empirisch bestimmt.

#### Umrechnung der Attribute

**age** Das Attribut **age** existiert sowohl im Zellulären Automaten als auch im Z-Format. Wir können es einfach übernehmen.

**familiarity** Auch das Attribut **familiarity** wird direkt übernommen.

**panic** Das Attribut **panic** existiert ebenfalls sowohl im Zellulären Automaten als auch im Z-Format. Seine Bedeutung ist jedoch unterschiedlich: Im Z-Format wird festgelegt, wie *anfällig* eine Person für Panik ist; im Zellulären Automaten gibt das Attribut hingegen an, wie panisch eine Person *zum aktuellen Zeitpunkt* ist. Wir initialisieren daher **panic** im Zellulären Automaten mit einem kleinen Wert (0,0001) und lassen das **panic**-Attribut aus dem Z-Format als Faktor in die Erhöhung der Panik im Zellulären Automaten einfließen, übernehmen also **panic** aus dem Z-Format als **panicFactor** im Zellulären Automaten.

**decisiveness / slackness** Das Attribut **decisiveness** aus dem Z-Format gibt an, wie entschieden sich eine Person verhält. Je entschiedener eine Person ist, desto weniger trödelt sie: Das `DefaultParameterSet` berechnet daher in der Methode `getSlacknessFromDecisiveness` das Attribut **slackness** für den Zellulären Automaten als $(1 - decisiveness) * 0,25$.

**exhaustion** Beim Attribut **exhaustion** verhält es sich ähnlich wie bei der **panic**: Im Zellulären Automaten gibt **exhaustion** an, wie erschöpft eine Person zum aktuellen Zeitpunkt ist. Dieser Wert wird mit Null initialisiert. Der **exhaustionFactor** bestimmt, wie schnell eine Person erschöpft, also wie gut bzw. schlecht ihre Kondition ist. Je höher der Faktor ist, desto schlechter ist die Kondition. Dies ist in unserer Simulation vom Alter abhängig. Im `DefaultParameterSet` wird daher in der Methode

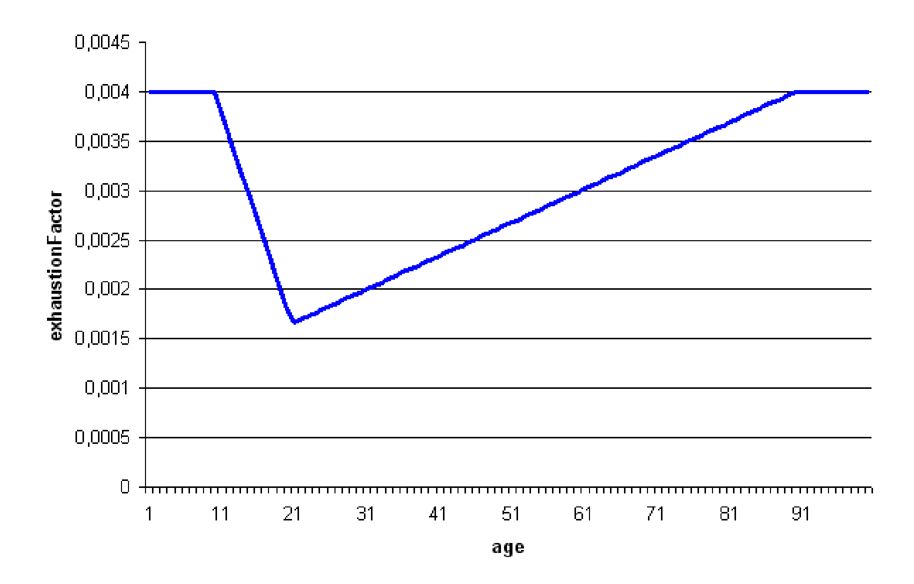



**Abbildung 6.3.:** Umrechnung des Attributs `age` in das Attribut `exhaustionFactor`

`getExhaustionFromAge` der **exhaustionFactor** aus dem Attribut **age** berechnet (siehe Abbildung 6.3).

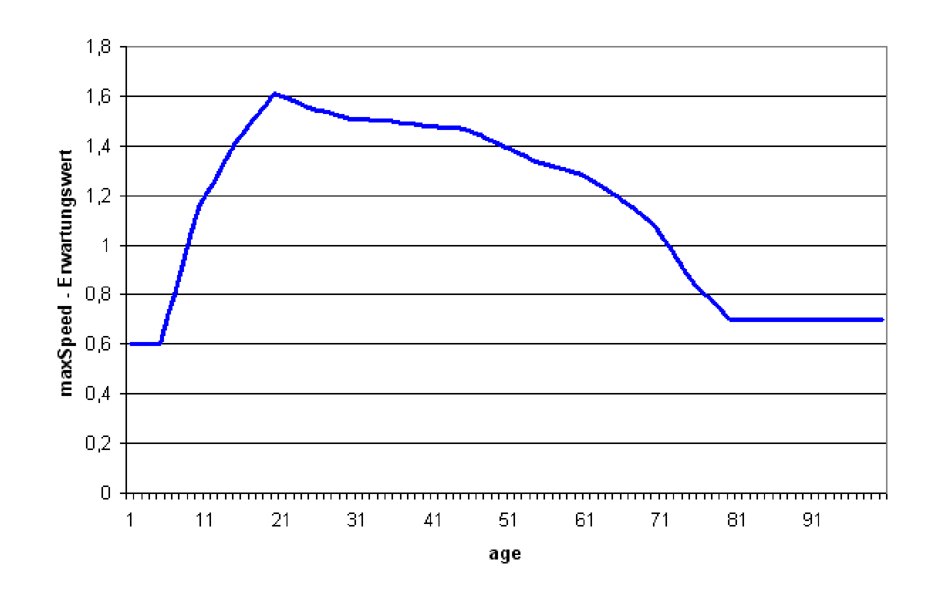



**Abbildung 6.4.:** Umrechnung vom Attribut age in den Erwartungswert für das Attribut maxSpeed (absolute Werte in $\frac{m}{sec}$)

**maxSpeed** Dieses Attribut gibt an, welche Geschwindigkeit die Person maximal erreichen kann. Dabei hat das Attribut einen Wert aus $[0, 1]$, da die Geschwindigkeit nur relativ zur einstellbaren absoluten maximalen Geschwindigkeit, die überhaupt irgendeine Person erreichen kann, angegeben wird. **MaxSpeed** ist von **age** abhängig und wird vom `DefaultPara-`

`meterSet` in der Methode `getSpeedFromAge` folgendermaßen berechnet: Zuerst wird ein vom Alter abhängiger Erwartungswert für **maxSpeed** berechnet (siehe Abbildung 6.4). Dieser Wert ist noch eine absolute Geschwindigkeit, also in Meter pro Sekunde angegeben und basiert auf einer empirischen Studie aus dem Rimea-Protokoll [1]. Der tatsächliche (absolute) Wert wird dann als Zufallswert aus einer Normalverteilung mit ebendiesem Erwartungswert, einer Varianz von $0,1$ und passendem Minimal- ($0,6\frac{m}{sec}$) und Maximalwert (absolute maximale Geschwindigkeit) bestimmt und in einen Wert relativ zur absoluten maximalen Geschwindigkeit umgewandelt. Ist dieser Wert noch größer als 1, wird er auf 1 heruntergesetzt.

**currentSpeed** Dieses Attribut gibt die jeweils aktuelle Wunschgeschwindigkeit des Individuums wieder. Es wird am Anfang auf **maxSpeed** initialisiert.

**reactiontime** Die Reaktionszeit einer Person ist die Zeit, die sie benötigt, um den Alarm zu bemerken und zu reagieren. Auch dieses Attribut ist von **age** abhängig und wird vom `DefaultParameterSet` in der Methode `getReactiontimeFromAge` durch folgende Formel berechnet:

$$reactiontime = \frac{age}{10}$$

Die Berechnung der Attribute hängt von der gewählten Implementierung des Interfaces `ParameterSet` ab. In der obigen Beschreibung wurde von der Benutzung des `DefaultParameterSets` ausgegangen, siehe dazu auch die ausführliche Erklärung in Abschnitt 7.1.7. Durch andere Implementierungen können die Abhängigkeiten aber (fast) beliebig angepasst werden.

## 6.4. Umwandlung für Graphen

Für die Konvertierung eines Z-Plans und seiner Belegung in eine passende Graphstruktur verwalten wir `NetworkFlowModel`-Objekte als Containerobjekte. Diese enthalten die **Fahrzeiten**, die **Kapazitäten** und die **Belegungen** für den Graphen, aber auch nützliche Zusatzinformationen zum Zeichnen des Graphen und ein `ZToGraphMapping`, das Abbildungen zwischen Räumen des Z-Plans und Knoten des Graphen und umgekehrt enthält. Durch die Verwendung

von Containerklassen muss bei der Organisation der Durchführung von Evakuierungen durch den Batchteil unseres Programms nur eine Klasse verwaltet werden.

Die Konvertierung selbst wird nicht intern im `NetworkFlowModel` durchgeführt, sondern in eine externe statische Konverter-Klasse `ZToGraphConverter` ausgelagert. Für diese Lösung spricht vor allem die Tatsache, dass der Konverter so nur seine schon vom Namen her eindeutige Aufgabe des Konvertierens übernehmen muss und dass man diesen leichter ergänzen oder sogar austauschen kann.

### 6.4.1. Modellierung

Bei der Umwandlung in den Graphen wird die beim Rastern erzeugte Gitterdarstellung verwendet. Es sind einige Modellierungsentscheidungen notwendig, die festlegen, wie der Graph aus dem Gitter entsteht.

#### Knoten

Die Knoten werden nicht, wie vielleicht auf den ersten Blick logisch und auch einfach erscheint, auf jeweils einzelne Rasterquadrate des gerasterten Plans gemappt.

Dies wäre zwar für den Vergleich späterer Ergebnisse mit denen des Zellulären Automaten vorteilhaft, einfache Laufzeituntersuchungen der von uns verwendeten Flussalgorithmen sprechen jedoch gegen eine solche direkte bijektive Abbildung zwischen Knoten und Rasterquadraten:

Wir verwenden in fast allen Flussalgorithmen (siehe Abschnitt 7.2) ein zeitexpandiertes Netzwerk. Die Knotenanzahl des zeitexpandierten Netzwerkes zu einem Graphen mit der Knotenmenge $V$ beträgt $|V_T| = \Omega(T \cdot |V|)$. In Anbetracht der Laufzeit des **Minimum Mean Cycle Canceling**-Algorithmus (MMCCA) von $\mathcal{O}(|V_T|^2 m^3 \log(|V_T|))$ (wobei $m = |E|$ die Anzahl der Kanten des zeitexpandierten Netzwerks ist, die ebenfalls um den Faktor $T$ größer ist als im Ursprungsnetzwerk), die die Gesamtlaufzeit der Graphalgorithmen dominiert, ist uns also daran gelegen, diesen Trade-Off zwischen Rechengenauigkeit und Laufzeit eher zugunsten einer geringeren Laufzeit zu entscheiden.

Für große Gebäude ist die Knotenanzahl sonst (vor allem bei hoher Genauigkeit bei der Fahrzeitberechnung und langem Zeithorizont) für eine praktisch durchführbare Berechnung zu groß.

**Algorithmus** Aus diesem Grund werden Rechtecke aus Rasterquadraten wie folgt zu Knoten zusammengefasst: Der Konvertierungsalgorithmus startet an einem Rasterquadrat und vergrößert die Dimension des aufspannenden Knoten-Rechtecks jeweils abwechselnd um einen Wert von Eins in eine der beiden Raum-Richtungen. Dies geschieht solange, bis eine eine Barriere den Weg zum nächsten Rasterquadrat versperrt, eine Raumwand erreicht wurde oder das nächste Kästchen zu einer anderen Art von Bereich gehört (Knoten sollen nur aus einem `Area`-Typ bestehen, was die nachfolgenden Berechnungen extrem vereinfacht. Zudem können unbetretbare Bereiche natürlich auch nicht zu einem Knoten gehören). Falls sich das Rechteck in eine Dimension nicht mehr vergrößern lässt, wird die andere Dimension erhöht, bis eine Balance-Differenz von $c$ (in GUI einstellbar, Standardwert ist Eins) zwischen beiden Dimensionen erreicht ist.
Auf diese Weise wird der gesamte Raum mit Knoten ausgefüllt. Tests haben akzeptable Knotenmengengrößen für verschiedene Räume ergeben.

### 6.4.2. Kanten

Kanten werden zwischen all jenen Knoten erzeugt, die über eine oder mehrere gemeinsame Rasterquadratseiten verfügen oder über Türen verbunden sind. Knoten, die nur einen Punkt, nicht jedoch eine Rasterquadratseite gemeinsam haben, sollen nicht über Kanten verbunden werden.

#### Fahrzeiten

Die Fahrzeiten zwischen den Knoten (sowohl zwischen Knoten innerhalb von Räumen, als auch zwischen verschiedenen Räumen) werden wie folgt berechnet:
Zunächst werden die Mittelpunkte beider Knotenrechtecke bestimmt, seien dies $A$ und $C$. Damit eine Kante zwischen zwei Knoten existiert, müssen diese entweder innerhalb eines Raumes gemeinsame Seitenkanten besitzen oder es

muss eine Tür zwischen beiden Knoten existieren. Der Mittelpunkt der gemeinsamen Tür- bzw. Seitenkante wird dann Punkt $B$. Die euklidische Länge des Streckenzugs $ABC$ entspricht dann der Fahrzeit.
Wie bereits bei der Knotenerzeugung erwähnt, spielt auch der Zeithorizont $T$ in zeitexpandierten Netzwerken eine große Rolle bei der Laufzeit. Aus diesem Grund haben wir uns entschieden, jede Fahrzeit mit einem global einstellbaren **precision**-Parameter zu gewichten, um den oben beschriebenen Trade-Off auch als Benutzer ein wenig steuern zu können. Die Fahrzeiten werden bei kleinerer Präzision verringert.
Desweiteren werden Knoten, die aus einer `DelayArea` bestehen (siehe Knotenerzeugung oben), noch mit dem **speedfactor** der entsprechenden `Area` multipliziert.

Bei Knoten, die Teil einer Treppe sind, müssen die Fahrzeiten ebenfalls angepasst werden werden. Im Gegensatz zu `DelayAreas` ist die Anpassung hier aber asymmetrisch: Wird zwischen zwei Knoten ein Höhenunterschied überbrückt, so muss die Fahrzeit in der einen Richtung anders angepasst werden als in der anderen. Hier wird wichtig, dass wir mit **gerichteten** Graphen arbeiten (was wir meistens zur Vereinfachung nicht erwähnen). Jede Kante in unserem Graphen existiert zweimal, einmal für jede der beiden möglichen Richtungen.

#### Kantenkapazitäten

Die Kapazität der Kanten ist proportional zur Länge der Kante, die die beiden nebeneinanderliegenden Polygone (Knoten) gemeinsam haben.

#### Knotenkapazitäten

Die Knotenkapazitäten entsprechen der Anzahl an Zellen, aus denen der Knoten besteht.

#### Belegungen

Da das `ConcreteAssignment` eine Abbildung zwischen einzelnen Personen und Koordinaten auf dem Raster liefert, muss nur für jede Person überprüft werden,

in welchem Knoten diese liegt und das entsprechende Assignment des Knotens für jede solche Person inkrementiert werden.

# 7. Verwendete Algorithmen

## 7.1. Das Verhalten des Zellulären Automaten

Unsere Simulation basiert auf dem Zellulären Automaten, den Klüpfel in [21] beschreibt. Wir haben diesen Automaten abgewandelt und beschreiben im folgenden Abschnitt ausführlich die von uns entwickelte Variante.

### 7.1.1. Grundlegende Struktur des Zellulären Automaten

#### Grundidee

Da es beim Zellulären Automaten beliebig viele Konfigurationsmöglichkeiten und Varianten des exakten Verhaltens gibt, haben wir uns dazu entschlossen, den Aufbau unseres Zellulären Automaten möglichst modular zu gestalten. Konkret haben wir uns dazu überlegt, dass das Verhalten des Zellulären Automaten durch **Übergangsregeln** gesteuert werden soll. Um die Ausführung der Regeln kümmert sich ein **Simulationsalgorithmus**. Die gewünschte Flexibilität erreichen wir nun zum einen dadurch, dass wir die Regeln, die für eine Simulation verwendet werden sollen, in einem **Regelsatz** zusammenfassen: Dieser stellt eine Menge von Regeln dar, welche in ihrer Gesamtheit eine komplette Konfiguration des Zellulären Automaten repräsentieren, so dass dessen genaues Verhalten über das Einbinden eines Regelsatzes definiert werden kann. Der zur Simulation verwendete Regelsatz kann zur Laufzeit ausgetauscht werden. Zum anderen verwenden wir einen Parametersatz, in dem alle von der Simulation verwendeten Parameter gespeichert werden. Dort werden auch Methoden zur Aktualisierung von Parametern oder Berechnungen von anderen

Werten (wobei diese Aktualisierungen/Berechnungen von Parametern abhängig sind) festgelegt. Auch der Parametersatz kann zur Laufzeit ausgetauscht werden.

#### Umsetzung in der Klassenstruktur

In unserem Programm wird ein Regelsatz durch die Klasse `RuleSet` repräsentiert, welche Regeln vom Obertyp `AbstractRule` enthält. Dabei erbt jede konkret verwendete Regel, die in den Regelsatz eingebunden werden soll, von der abstrakten Klasse `AbstractRule`. Dadurch wird sichergestellt, dass alle Anforderungen an eine konkrete Regel erfüllt werden. In erster Linie bedeutet das hier, dass jede Regel die Methoden `executableOn()` und `onExecute()` implementieren muss. Die Methode `executableOn()` überprüft, ob die Vorbedingungen zur Anwendung einer Regel gegeben sind, während die Methode `onExecute()` die Semantik einer Regel definiert. Für die konkreten Regeln verwenden wir eine Vielzahl von Untertypen von `AbstractRule`, welche wir in Abschnitt 7.1.3 vorstellen. Parametersätze werden von der Klasse `ParameterSet` realisiert.

### 7.1.2. Unser Simulationsalgorithmus

Ein Simulationsalgorithmus erbt von der abstrakten Klasse `EvacuationCellularAutomaton` und beginnt mit der Initialisierung in der Methode `initialize()`. Dort werden die statischen Datenstrukturen initialisiert und die Controller-Objekte für Potential und Statistik initialisiert. Aus dem global verfügbaren `PropertyContainer` werden das `ParameterSet` und das `RuleSet`, die benutzt werden sollen, geladen.

Die Ausführung beginnt mit dem Aufruf von `run()`. Zuerst werden die Initialisierungsregeln aus dem `RuleSet` ausgeführt (die Methode `initialize()` initialisiert nur die Datenstrukturen des Rahmenalgorithmus, die von der speziellen Variante des Zellulären Automaten unabhängig sind). Dann werden in einer Schleife die Abbruchbedingungen überprüft und für jeden Zeitschritt die Methode `executeStep()` aufgerufen.

Unser Zellulärer Automat realisiert eine **zeitdiskrete** Simulation, d.h. die Simulation schreitet in festen Zeitschritten voran. In jedem Simulationsschritt iteriert der Simulationsalgorithmus über alle Individuen, die sich noch in der Simulation befinden. Für eine Zelle $Z$, die von einem Individuum besetzt wird, versucht der Algorithmus alle (Schleifen-)Regeln aus dem gewählten Regelsatz auf $Z$ anzuwenden [1]. Ob eine Regel auf eine Zelle $Z$ angewendet werden kann, entscheidet dabei die Methode `executableOn()`, die von jeder Regel implementiert werden muss.

Nach Anwendung aller Regeln in diesem Zeitschritt werden dann noch alle als „zu entfernen“ markierten Individuen auch wirklich aus der Simulation entfernt (warum dies nicht in den Regeln selbst passiert, siehe Abschnitt *Konkrete Evakuierungsregeln* in Abschnitt 7.1.6), und eine Aktualisierung des dynamischen Potentials wird durchgeführt (wie im Abschnitt *Dynamisches Potential* in Abschnitt 7.1.4 beschrieben). Danach werden wieder die Abbruchbedingungen geprüft und ggf. wieder `executeStep()` für den nächsten Zeitschritt aufgerufen.

Ein Simulationslauf endet, wenn entweder alle Individuen evakuiert worden sind oder wenn eine maximale Zeit abgelaufen ist (siehe auch *Ende der Simulation* in Abschnitt 7.1.5). Gegebenenfalls kann eine Simulation auch schrittweise ablaufen.

Die vorhandenen Simulationsalgorithmen lassen sich in zwei Typen einteilen: Algorithmen, die auf dem zellulären Automaten in reiner Form arbeiten und Algorithmen, die die Ergebnisse eines Fluchtplanes nutzen. Jeden Typ gibt es in den gleichen drei Ausführungen:

**Einheitsreihenfolgeautomat** Der einfachste Automat fragt die Individuen in jeder Runde in derselben Reihenfolge ab. Dies kann in einer zu niedrigen Evakuierungszeit resultieren, wenn sich die Personen z.B. in einem engen Gang nie gegenseitig behindern. Dieser Automat wird von der Klasse `CellularAutomatonInOrderExecution` realisiert.

**Realzeitsimulationsautomat** Einen Sonderfall stellt der in der Klasse `CARealTime` implementierte Zelluläre Automat dar. Er ist ein Einheitsreihen-

[1] Obwohl sowohl die Ausführungsreihenfolge der Regeln fest vorgegeben ist, als auch immer alle ausführbaren Regeln ausgeführt werden, kann eine zufällige Regelauswahl und -ausführung implementiert werden: Dazu muss nur ein neuer Regelsatz implementiert und dem Simulationsalgorithmus als Parameter übergeben werden.

folgeautomat, der eine Pause zwischen den Schritten durchführt, die mit `setStepTime` gesetzt werden kann. Damit erlaubt er eine Realzeitsimulation bzw. Visualisierung, falls die Simulation schnell genug ist.

**Zufallsreihenfolgeautomat** Im Gegensatz zu den beiden vorigen Automaten führen die Personen hier in jeder Runde ihre Aktionen in zufälliger Reihenfolge aus. Dies realisiert einen faireren Ablauf. Deshalb ist dieser Automat auch der Standardautomat. Das Vermischen der Individuen benötigt jedoch zusätzliche Laufzeit. Realisiert ist dieser Automat in der Klasse `CellularAutomatonRandomOrderExecution`.

**Quetschregelautomat** Dieser Automat ist ein Spezialfall des Zufallsreihenfolgeautomaten. Er ermöglicht, dass zwei Personen auf benachbarten Zellen ihre Positionen tauschen, indem sie sich aneinander „vorbeiquetschen“. So können Blockadesituationen in engen Gängen bzw. sehr vollen Räumen aufgelöst werden; für die Simulation der optimierten Fluchtpläne wird dieser Automat empfohlen.

Die Simulation eines Schrittes mit diesem Automaten ist komplizierter als in den anderen Varianten. Zunächst wird eine Test-Simulationsrunde durchgeführt, dabei melden alle Individuen die Zellen, die sie gern betreten möchten. Anschließend wird getestet, ob zwei Individuen freiwillig ihre Positionen tauschen möchten.

Die Positionen zweier Individuen werden nur getauscht, wenn die jeweiligen Zellen von beiden mit höchster Priorität als Zielzelle angegeben werden. Positionstausch mit mehr als zwei Personen wird nicht unterstützt.

Da der Tausch in mehreren Schritten abläuft, ist dieser Automat deutlich langsamer als die anderen. Er wird in der Klasse `SwapCellularAutomaton` implementiert.

### 7.1.3. Die Regeln

#### Regeltypen

Da der Simulationsalgorithmus *alle* Regeln des Regelsatzes ausführt, müssen wir sicherstellen, dass sie zueinander kompatibel sind und zusammen ein voll-

ständiges Regelwerk zur Steuerung des Zellulären Automaten ergeben. Um dies übersichtlicher zu gestalten, klassifizieren wir die Regeln gemäß ihres Einsatzgebietes und fordern anschließend, dass es in jedem Regelsatz zu jedem Einsatzgebiet *mindestens* eine Regel geben muss. Damit sichern wir die Vollständigkeit des Regelsatzes. Andererseits darf es für bestimmte Einsatzgebiete *nicht mehr* als eine Regel geben, damit es nicht zu Inkompatibilitäten kommt. Konkret teilen wir unsere Regeln gemäß folgender Einsatzgebiete auf:

**Bewegungsregeln** Bewegungsregeln versetzen ein Individuum von einer Zelle auf eine andere und aktualisieren dabei ggf. die dynamischen Parameter des Individuums (mit Hilfe des ParameterSets). Sie sind nur auf Zellen anwendbar, auf denen auch ein Indiviuum steht. Es darf nur eine Bewegungsregel pro Regelsatz geben. Bewegungsregeln erben von der Klasse `AbstractMovementRule`.

**Potentialregeln** Die Potentialregeln initialisieren die statischen Potentiale zu Beginn der Simulation und können einem Individuum ein neues statisches Potential zuweisen. Konkret wird zum Initialisieren der Potentiale eine Regel des Typs `AbstractInitialPotentialRule` benötigt. In dieser sollte überprüft werden, ob überhaupt alle Individuen evakuiert werden können – möglicherweise führt für einige Individuen von ihrem Aufenthaltsort überhaupt kein Weg zu einem Ausgang. Diese werden sofort entfernt, um eine nicht-terminierende Simulation zu verhindern. Regeln, die Potentiale initialisieren, sind nur zu Beginn der Simulation ausführbar.

Damit Individuen während des Ablaufs der Simulation ihr statisches Potential ändern können, hat der Benutzer die Möglichkeit, eine oder mehrere Regeln vom Typ `AbstractPotentialChangeRule` zu laden.

**Evakuierungsregeln** Evakuierungsregeln gliedern sich in zwei Untertypen: **Regeln für das Evakuieren von Individuen**, welche Individuen aus der Simulation entfernen, die einen Evakuierungsbereich betreten haben, und **Regeln für das Retten von Individuen**, welche aktiv werden, wenn ein Individuum einen Bereich betritt, in dem es gerettet ist. Wir verwenden für die beiden Untertypen die Klassen `AbstractEvacuationRule` bzw. `AbstractSaveRule`.

**Reaktionsregeln** Reaktionsregeln sorgen dafür, dass jedes Individuum nach

Ablauf seiner Reaktionszeit seine Flucht beginnt. Sie erben von der Klasse `AbstractReactionRule`.

Jedes Einsatzgebiet besitzt also mindestens eine abstrakte Klasse. Diese erbt stets von der Klasse `AbstractRule`.

#### Definieren der Semantik

Die Semantik einer Regel wird, wie oben beschrieben, in der Methode `onExecute()` implementiert. Die Methode wird nur ausgeführt, wenn die Regel auf die angegebene Zelle anwendbar ist. Hier kann beliebiger Code implementiert werden, um den Zellulären Automaten zu steuern. Dabei stehen alle Methoden, die im Datenmodell des Zellulären Automaten implementiert sind, zur Verfügung, weil die abstrakte Klasse `AbstractRule`, von der jede konkrete Regel erben muss, einen `CAController` verwaltet, über den man an das Datenmodell des Zellulären Automaten gelangen kann.

#### Initialisierungs- und Schleifenregeln

Wie im Abschnitt 7.1.2 beschrieben, werden zuerst ein einziges Mal alle Initialisierungsregeln auf alle Individuen angewendet und dann für jeden Schritt jeweils alle Schleifenregeln. Ob eine Regel eine Initialisierungsregel oder eine Schleifenregel ist, ist keine Eigenschaft der Klasse `AbstractRule`. Dies sind nur die beiden Gruppen von Regeln, aus denen das `RuleSet` besteht. So kann man eine Regel z.B. auch in beide Gruppen einfügen, wenn sie sowohl vor dem ersten Zeitschritt als auch in jedem Zeitschritt ausgeführt werden soll. Natürlich ist es bei manchen Regeln, wie z.B. denen vom Typ `AbstractInitialPotentialRule` wenig sinnvoll, sie auch als Schleifenregel auszuführen.

#### Standardregeln

Eine Beschreibung der voreingestellten konkreten Regeln findet man weiter unten in Abschnitt 7.1.6.

### 7.1.4. Realisierung der Potentiale

Bei der Gestaltung eines Zellulären Automaten ist es hilfreich, Informationen über die Richtung, die ein Individuum nehmen muss, wenn es auf dem kürzesten Weg zu einem Ausgang gelangen will, bereitzustellen. In unserem Modell wird dies über **statische Potentiale** realisiert. Weiterhin ist es wünschenswert, ein Gruppenverhalten (den „Herdentrieb“) zu modellieren. Dies geschieht über **dynamische Potentiale**. Beiden Fällen ist gemein, dass jeder Zelle des Automaten ein Wert, der sogenannte Potentialwert, zugeordnet wird. Je nach Fall wird der Wert allerdings unterschiedlich berechnet und interpretiert.

#### Statisches Potential

Die Potentialwerte der Zellen bei einem statischen Potential repräsentieren den Abstand dieser Zelle zu dem Ausgang, zu dem das Potential gehört. Man kann sich also – wenn man die Potentialwerte als Höhenwerte interpretiert – ein Gefälle vorstellen, das zu den Ausgängen hin abfällt. Passend zu dieser Interpretation bewegen sich Individuen in Richtung kleinerer (statischer) Potentialwerte, also bergab.

Als Ausgang wird jeweils ein zusammenhängender Block von Exitzellen verwendet. Für jeden solchen Ausgang wird ein statisches Potential berechnet:

Die Exitzellen des gewählten Blocks bekommen den Potentialwert 0. Die restlichen Potentiale werden nach folgendem Algorithmus bestimmt:

1. Fülle eine Warteschlange $Q$ mit den Exitzellen des gewählten Blocks.

2. Durchlaufe $Q$ und fülle eine „Kinder-Warteschlange“ $C$ mit Nachbarzellen der Zellen aus $Q$, die noch nicht behandelt wurden. Hierbei werden Tupel der Form (Zelle $z$, Abstandswert $d$, Anzahl Eltern $p$, Summe der Elternwerte $s$) für jede Kindzelle angelegt. Dabei ist die Zelle eine Referenz auf die Kindzelle, der Abstandswert ist das Minimum der Abstandswerte aller Eltern, Anzahl Eltern erhöht sich bei jedem Elter, der diese Zelle berührt, um 1, und die Summe entsprechend des Wertes des Elters.

3. Ist $C$ leer, so stoppe.

4. Durchlaufe die Zellen in $C$ und wende **Smoothing** auf den Potentialwert jeder Zelle nach folgender Formel an: $d_{kind} = \frac{3d+s}{3+p}$.
5. Setze $Q = C$ und gehe zu 2.

So verfährt man mit allen zusammenhängenden Blöcken von Exitzellen und bekommt *je ein* statisches Potential für *einen* Ausgang.

### Dynamisches Potential

Diese Art von Bodenfeld dient dazu, Interaktionen zwischen einzelnen Individuen zu modellieren. Genauer soll ein Herdenverhalten simuliert werden: ein Individuum hat eine gewisse Tendenz, anderen Individuen zu folgen, anstatt selber einen Weg zu suchen. Um dies zu realisieren hinterlassen Individuen „virtuelle Spuren“ auf dem dynamischen Potential – es bekommt dort einen (echt) positiven Wert, wo kurz zuvor ein Individuum war. Dies passiert in der Bewegungsregel.

In jedem Zeitschritt des Zellulären Automaten wird dann eine Aktualisierung des dynamischen Potentials durchgeführt: Zwei Parameter, $\alpha$ und $\delta$, steuern die Entwicklung des Feldes. Mit Wahrscheinlichkeit $\delta$ sinkt der dynamische Potentialwert einer Zelle um 1 (aber nie unter 0), mit Wahrscheinlichkeit $\alpha$ wird der dynamische Potentialwert einer Nachbarzelle einer Zelle, die einen positiven dynamischen Potentialwert hat, um 1 erhöht. Mit $\delta$ kann somit die Länge der Spur, die ein Individuum hinter sich herzieht, gesteuert werden. $\alpha$ reguliert die Ausweitung dieser Spur.
Die Parameter $\alpha$ und $\delta$ werden aus den Gewichtungsparametern **probabilityDynamicIncrease** und **probabilityDynamicDecrease** aus dem ParameterSet ausgelesen.

Auf dem dynamischen Bodenfeld bedeutet also ein höherer Potentialwert eine höhere Attraktivität (im Gegensatz zum statischen Potential, bei dem kleinere Werte attraktiver sind).

### Verwendung der Potentiale durch Individuen

In die Bewegung der Individuen gehen beide Potenziale gewichtet ein, dazu werden **Coupling-Konstanten** $k_S$ und $k_D$ benutzt, die die Gewichtung des

statischen und dynamischen Potenzials angeben. Je niedriger die Werte sind, desto geringer ist der Einfluss des entsprechenden Potenzials. Hierbei kann $k_S$ als Maß für die Ortskenntnis aufgefasst werden ($k_S = 0$ würde dann völlige Unkenntnis bedeuten). $k_D$ kann als Tendenz aufgefasst werden, anderen Personen zu folgen.

Da im Allgemeinen die Wahl der Parameter schwierig ist und erheblichen Kalibrierungsaufwand erfordert und der Zelluläre Automat weitere Parameter, wie z.B. Panik, in die Bewegung einfließen lässt, können diese Parameter in einem `ParameterSet` und in speziellen Eigenschaftendateien angepasst werden. Damit können leicht unterschiedliche Verahltensweisen mit einem Bewegungsalgorithmus realisiert werden.

#### Implementierung

Die beschriebene Funktionalität wird in der Klasse `SPPotentialController` realisiert. Diese Klasse implementiert das Interface `PotentialsController`. Somit ist es möglich, relativ einfach andere Potentialberechnungsfunktionen, aber auch ein anderes Verhalten des dynamischen Bodenfeldes einzubinden.

### 7.1.5. Rettung der Individuen

#### Gerettete Individuen

Sobald ein Individuum eine `Save-` oder `ExitCell` erreicht hat, gilt es als evakuiert. Das Individuum bekommt ein neues Potential zugewiesen (siehe nächster Abschnitt). Wenn das Individuum gerade eine `ExitCell` betreten hat, wird es am Ende des Zeitschrittes aus der Simulation entfernt. (Siehe hierzu auch *Konkrete Evakuierungsregeln* in Abschnitt 7.1.6.)

#### Neues Potential für gerettete Individuen

Individuen, die einen sicheren Bereich betreten, sind schon gerettet, sollen allerdings weiterlaufen, bis sie einen Ausgangsbereich erreichen. Damit Individuen in sicheren Bereichen sich nicht wieder aus diesen Bereichen herausbewegen,

bekommen sie bei Betreten eines sicheren Bereichs ein neues Potential zugewiesen. Sonst könnte es passieren, dass ein Individuum einen sicheren Bereich verlassen möchte um zu einem anderen Ausgang (als dem an den Sicherheitsbereich angrenzenden) zu laufen.
Mit einem ähnlichen Algorithmus wie zur Erzeugung der Potentiale werden beim Konvertieren des Z-Formats in einen Zellulären Automaten zu jedem Block aus `ExitCells` die angrenzenden Blöcke aus `SaveCells` berechnet. Dies macht die Methode `saveCellSearch()` der Klasse `ZToCAConverter`. Der Algorithmus sucht ausgehend vom Block der Evakuierungszellen alle angrenzenden Sicherheitszellen und speichert die Zellen in einer Warteschlange. Diese werden dann der Reihe nach abgearbeitet, dabei werden weitere erreichbare Sicherheitszellen in die Warteschlange eingefügt. Damit keine Zelle doppelt betrachtet wird, werden die bereits gefundenen Zellen markiert.
Jede `SaveCell`, die von einem `ExitCell`-Block über einen Pfad von `SaveCells` erreichbar ist, bekommt als Attribut das Potential, das zu dem `ExitCell`-Block gehört, zugewiesen. Sind von einer `SaveCell` mehrere Ausgänge zu erreichen, bekommt die `SaveCell` das Potential des `ExitCell`-Blocks, der in kleinster Entfernung liegt, zugewiesen. Dieses Potential wird dann zum neuen Potential eines jeden Individuums, das die entsprechende `SaveCell` betritt, so dass das Individuum zum passenden Ausgang läuft. Zusätzlich wird in den implementierten Bewegungsregeln dafür gesorgt, dass ein Individuum einen `SaveCell`-Block nicht mehr verlassen kann, außer zu einem angrenzenden `ExitCell`-Block.
Für `SaveCell`-Blöcke, die nicht von einem `ExitCell`-Block erreichbar sind, (die z.B. einen begrenzt großen Bunker simulieren), gilt eine Sonderregelung. Hier bekommen die Individuen, die diesen Bereich betreten, ein Potential zugewiesen, bei dem alle Zellen des sicheren Bereiches gleiche Potentialwerte haben und alle Zellen, die außerhalb liegen, keinen Potentialwert. Damit bewegen sich die Individuen mit keinem bestimmten Ziel (d.h. sie blockieren keinen Randbereich des sicheren Bereichs) und sie verlassen den Bereich nicht.

### Nicht gerettete Individuen

Es gibt zwei Situationen, in denen Individuen nicht gerettet werden können:

**Kein Ausgang erreichbar** Wenn ein Individuum zu Beginn der Simulation an

einer Stelle im Gebäude steht, von der aus kein Ausgang erreichbar ist, es also praktisch eingeschlossen ist, wird das Individuum entfernt und als *nicht gerettet* gezählt. Die Entfernung des Individuums geschieht gleich zu Beginn in der `InitialPotentialConcreteRule`, denn diesen Individuen kann ja gar kein Potential zugewiesen werden.

**Die Zeit reicht nicht** Wenn nach Ablauf der vom Benutzer eingegebenen maximalen Simulationszeit ein Individuum noch keine `Save-` oder `ExitCell` erreicht hat, gilt es als nicht gerettet. Nach Ablauf der Simulation werden alle Individuen, die noch nicht evakuiert sind, als *nicht gerettet* markiert und aus der Simulation entfernt.

#### Ende der Simulation

Die Simulation endet, wenn für alle Individuen gilt, dass sie entweder *evakuiert* oder *nicht gerettet* sind. Evakuiert bedeutet dabei, dass ein Individuum sich auf einer `SaveCell` oder `ExitCell` befindet (oder sich auf einer `ExitCell` befand und dann mit *evakuiert* markiert aus der Simulation entfernt wurde). Die Simulation endet spätestens, nachdem die vom Benutzer eingegebene maximale Simulationszeit abgelaufen ist, denn dann werden ja alle noch nicht evakuierten Individuen als *nicht gerettet* markiert.

### 7.1.6. Voreingestellte konkrete Regeln

Im Folgenden werden die voreingestellten Regeln des `DefaultRuleSet` beschrieben und auch die Regeln, die man alternativ dazu einstellen kann. Natürlich können auch noch weitere Regeln implementiert werden.

#### Konkrete Bewegungsregeln

Wir haben zwei Bewegungsregeln implementiert, die Nicht-Warteregel (`NonWaitingMovementRule`) und die Warteregel (`WaitingMovementRule`). Sie unterscheiden sich darin, dass Individuen in der Warteregel freiwillig mit gewisser Wahrscheinlichkeit auf ihrer Position stehen bleiben, während sie bei der Nicht-Warteregel lieber einen Schritt zurück gehen. Dies führt dazu, dass

bei Stauungen Personen, die hinten warten müssen, abwechselnd einen Schritt vor und zurückgehen.

Wenn eine der beiden Bewegungsregeln ausgeführt wird, passiert Folgendes (eine Übersicht dieses Ablaufs findet man unterhalb dieses Textes):
Zunächst wird geprüft, ob das Individuum überhaupt alarmiert ist. Falls nicht, passiert nichts; andernfalls wird zuerst geprüft, ob das Individuum mit seiner letzten Bewegung fertig ist. Falls es das nicht ist, passiert ebenfalls nichts. Falls es mit der letzten Bewegung fertig ist, wird getestet, ob das Individuum slackt, d.h. trödelt. Falls es das tut, werden nur Erschöpfung und Wunschgeschwindigkeit (für die nächste Bewegung) des Individuums aktualisiert.

Falls das Individuum nicht trödelt, wird eine Zielzelle ausgewählt, die Bewegung durchgeführt und als letztes die Wunschgeschwindigkeit für den nächsten Schritt aktualisiert.

Zur Wahl der Zielzelle werden die freien Nachbarzellen gemäß ihres Potentialwertes (sowohl statisch als auch dynamisch) sortiert und mit entsprechender Wahrscheinlichkeit ausgewählt. Die genaue Gewichtung bei der Auswahl wird vom gewählten `ParameterSet` festgelegt. Die Wahrscheinlichkeit, weiter in die bisherige Laufrichtung zu gehen, wird etwas erhöht.

Nach der Wahl der Zielzelle werden Erschöpfung und Panik aufgrund dieser Wahl aktualisiert. Nun wird festgestellt, ob das Individuum vielleicht doch stehenbleibt (d.h. falls Zielzelle = aktuelle Zelle des Individuums). Gründe dafür können sein, dass alle Zellen um das Individuum herum besetzt sind, dass das Individuum lieber stehenbleiben möchte als einen Schritt zurückzugehen, oder dass das Individuum versucht hat, einen sicheren Bereich zu verlassen, was ihm aber verboten wird (in letzterem Fall wurde die Panik nicht erhöht, obwohl das Individuum seine Zielzelle nicht betreten darf).

Wenn das Individuum sich tatsächlich bewegen soll, wird zunächst der Wert des dynamischen Potentials der Zelle, auf dem das Individuum noch steht, erhöht. Dann wird seine wirkliche Geschwindigkeit bestimmt, die seiner Wunschgeschwindigkeit multipliziert mit den Geschwindigkeitsfaktoren der Verzögerungs- und Treppenbereiche, über die das Individuum gerade vielleicht läuft, entspricht.

Bei der Realisierung der Geschwindigkeiten wird es etwas knifflig. Wenn Individuen sich immer zellenweise bewegen, können nur sehr wenige Geschwindigkeiten umgesetzt werden (Vielfache der Zeit, die man für die Überquerung einer Zelle benötigt). Stattdessen verwalten wir die Geschwindigkeiten intern kontinuierlich: Jedes Individuum speichert die Zeit, zu der es mit der aktuellen Bewegung fertig ist; diese Zeit kann über `getStepEndTime()` abgefragt werden. Bevor diese Zeit nicht überschritten worden ist, kann sich das Individuum nicht mehr bewegen. Wenn sich das Individuum dann tatsächlich bewegt, wird die Zeit, die es mit der berechneten wirklichen Geschwindigkeit benötigt, um die Distanz (entweder 0,4 oder 0,57 Meter) zurückzulegen, berechnet und zur letzten `StepEndTime` hinzuaddiert. Das Individuum wird nun direkt auf die Zielzelle gesetzt.

Die Bewegung läuft also abschnittsweise: ein Individuum bewegt sich in einem Schritt in Nullzeit von einer Zelle zur nächsten und bleibt dort eine gewisse Zeit stehen, so dass im Durchschnitt die gewünschte Geschwindigkeit herauskommt. Die kontinuierlichen Zeitpunkte der Bewegung werden gespeichert, so dass sie zur flüssigen Visualisierung benutzt werden können. Aus diesem Grund wird auch zu Beginn der Bewegung getestet, ob das Individuum seine letzte Bewegung bereits abgeschlossen hat.

#### Übersicht: Der Ablauf der `(Non-)WaitingMovementRule`

```
1  Regel bewegeIndividuum(Individuum i)
2  Falls i noch nicht alarmiert ist:
3     Tue nichts
4  Sonst:
5     Falls i noch vorherige Bewegung ausführt:
6        Tue nichts
7     Sonst:
8        Falls i trödelt:
9           Aktualisiere Erschöpfung
10          Aktualisiere Wunschgeschwindigkeit
11       Sonst:
12          Bestimme Zielzelle
13          Aktualisiere Panik und Erschöpfung
14          Falls i stehenbleibt:
```

```
15              Aktualisiere Wunschgeschwindigkeit
16          Sonst:
17              Erhöhe dynamischen Potentialwert
18              Berechne wirkliche Geschwindigkeit
19              Berechne neue StepEndTime
20              Setze i auf Zielzelle
21              Aktualisiere Wunschgeschwindigkeit
```

### Konkrete Potentialregeln

Wir unterscheiden zwei Typen von Potentialfeldern (auch Potentiale oder Bodenfelder genannt): statische und dynamische Potentiale. Die Berechnung der statischen Potentiale und die Aktualisierung des dynamischen Potentials wurden bereits in Abschnitt 7.1.4 beschrieben. Hier wollen wir noch erklären, wie Individuen ihr anfängliches statisches Potential erhalten und auf welche Weise sie dieses während der Simulation wechseln. Die Initialisierung des dynamischen Bodenfelds der Individuen besteht nur darin, jedem Individuum das (einzige) dynamische Potential zuzuweisen. Dieses wird von der `InitialDynamicPotentialConcreteRule` geleistet.

Die initiale Potentialregel für statische Bodenfelder muss etwas mehr leisten. Wir haben das gewünschte Verhalten in der `InitialPotentialConcreteRule` implementiert. Diese prüft zunächst für jedes Individuum, ob es überhaupt ein Potential gibt, welches das jeweilige Individuum zu einem Ausgang führen kann. Ein solches Potentialfeld nennen wir im Folgenden ein für dieses Individuum **wählbares** Potential. Falls für ein Individuum kein wählbares Potentialfeld existiert, gibt es für das betreffende Individuum auch keine Möglichkeit, zu einem Ausgang zu gelangen. Deshalb wird es als *nicht gerettet* markiert und muss während des weiteren Verlaufs der Evakuierungssimulation nicht mehr betrachtet werden.

Wenn für ein Individuum mindestens ein wählbares Potentialfeld existiert, wählen wir in Abhängigkeit vom `familiarity`-Wert des Individuums eines der wählbaren Potentiale aus. Dazu wird zunächst eine Vorauswahl aus den Potentialfeldern getroffen: Alle wählbaren Potentialfelder werden (aufsteigend) nach der Weglänge des Weges sortiert, den das Individuum zurücklegen muss, wenn es über das entsprechende Potentialfeld zum Ausgang läuft. Dann werden die

$k$ besten Potentiale ausgewählt, wobei $k$ vom `familiarity`-Wert abhängt und bei größerer Ortskenntnis kleiner ist. Aus der Vorauswahl von $k$ Potentialen wird dann das Potential als statisches Potential für das Individuum gewählt, dessen Ausgang den höchsten Attraktivitätswert hat (die Attraktivität eines Ausgangs kann im Z-Editor angegeben werden).

Durch das Vorgehen erreichen wir, dass ortskundige Menschen entlegene Ausgänge nicht in Betracht ziehen, während ortsunkundige Personen keine solchen Informationen ausnutzen können. Das ist naheliegend, weil jemand, der sich gut auskennt, mit sehr hoher Wahrscheinlichkeit einen Fluchtweg wählt, der in kürzester Distanz liegt (und die Person der Erwartung nach möglichst schnell in Sicherheit bringt). Wenn sich jemand nicht auskennt, besteht immerhin noch die Möglichkeit, zufällig einen guten Ausgang zu kennen, ebensogut kann es aber sein, dass derjenige zu einem weit entfernten Ausgang läuft.

Aus der Menge der Ausgänge, die eine Person als möglichen Ausgang erwägt, wird dann anhand der Attraktivität dieses Ausgangs entschieden, weil Menschen dazu tendieren, attraktivere Ausgänge zu wählen.

Für das Wechseln der statischen Potentialfelder während der Evakuierungssimulation stehen insgesamt folgende Regeln zur Verfügung, die in den nachfolgenden Absätzen genauer erläutert werden:

- ChangePotentialAttractivityOfExitRule
- ChangePotentialFamiliarityRule
- ChangePotentialFamiliarityOrAttractivityOfExitRule

Alle drei Regeln verwenden die Methode `changePotentialThreshold`, um zu entscheiden, ob das Individuum aufgrund seiner Panik das Potential wechseln möchte (Die Methoden unseres `DefaultParameterSets` werden in Abschnitt 7.1.7 beschrieben). Nur wenn das Bodenfeld tatsächlich gewechselt werden soll, werden die Regeln wie im Folgenden beschrieben ausgeführt.

Die `ChangePotentialAttractivityOfExitRule` wählt das neue Potential eines Individuums nur nach der Attraktivität: Dem Individuum, das sein Potentialfeld wechseln soll, wird der Ausgang mit der höchsten Attraktivität zugewiesen. Wenn beim Wechsel des Potentials ausschließlich diese Regel verwendet wird, flüchten immer mehr Individuen durch denselben Ausgang (den Ausgang mit der höchsten Attraktivität).

Die `ChangePotentialFamiliarityRule` richtet sich hingegen nach der Ortskenntnis der Personen. Wie bei der Initialisierung werden zunächst die besten $k$ Potentiale ausgewählt, wobei $k$ vom `familiarity`-Wert des Individuums abhängt. Anders als bei der Initialisierung wird aus dieser Vorauswahl aber *nicht* anhand der Attraktivität entschieden, sondern uniform zufällig ausgewählt (andernfalls würde das Potential ja auch eventuell gar nicht wirklich gewechselt).

Die `ChangePotentialFamiliarityOrAttractivityOfExitRule` stellt eine Kombination aus den beiden Regeln `ChangePotentialAttractivityOfExitRule` und `ChangePotentialFamiliarityRule` dar. Mit der Wahrscheinlichkeit `probabilityChangePotentialFamiliarityOrAttractivityOfExitRule` wird die `ChangePotentialFamiliarityRule` gewählt, mit der entsprechenden Gegenwahrscheinlichkeit wird die `ChangePotentialAttractivityOfExitRule` gewählt.

Neben den bereits beschriebenen Regeln verwenden wir noch eine weitere Regel, die mit Potentialwechseln zusammenhängt. Dabei geht es darum, dass Individuen ihr Potential nicht nur aufgrund von Panik wechseln, sondern evtl. auch dann einen anderen Ausgang wählen, wenn sie mit dem Vorwärtskommen unzufrieden sind (z.B. weil der Weg zum gewählten Ausgang verstopft ist). Für diesen Fall haben wir die `ChangePotentialInsufficientAdvancementRule` implementiert. Sie erlaubt den Individuen, ihr Potential zu wechseln, wenn sie während einer bestimmten Anzahl an Schritten des Zellulären Automaten nicht genügend Zellen auf den vom gewählten Potential bestimmten Ausgang zugelaufen sind (was man durch die überbrückte Potentialdifferenz messen kann). Die Regel verwendet einen Zähler, der während der Simulation herabgezählt wird. Bei Initialisierung des Zählers wird das aktuelle Potential gespeichert – ist das Potential nach Ablauf des Zählens nicht wesentlich niedriger, so darf das Individuum sein Potential wechseln. Als neues Potential wird das nächstbeste ausgewählt. Die Regel berücksichtigt außerdem, dass langsamere Individuen weniger Geduld haben, da sich eine Stauung auf diese Individuen noch deutlicher auswirkt (eine zu große Verzögerung kann dann eher das Erreichen des Ausgangs verhindern). Langsamere Individuen dürfen deshalb eher wechseln als schnellere Individuen.

### Konkrete Evakuierungsregeln

Im Bereich der **Regeln für das Retten von Individuen** (`AbstractSaveRule`) haben wir die `SaveIndividualsRule` implementiert. Sie wird sowohl ausgeführt, wenn sich ein Individuum auf eine `ExitCell` bewegt als auch, wenn es sich auf eine `SaveCell` bewegt, da beide Zellarten zu Bereichen gehören, in denen Individuen gerettet sind. Falls das Individuum nicht schon als gerettet markiert ist, passiert Folgendes:

- Das Individuum wird als gerettet markiert.
- Die zur Evakuierung des Individuums benötigte Zeit wird gespeichert.
- Das Attribut **panic** wird auf 0 gesetzt (und später auch nicht wieder erhöht).
- Falls die betretene Zelle eine `SaveCell` ist, bekommt das Individuum ein neues Potential zugewiesen, das es zum richtigen Ausgang führt (siehe *Neues Potential für gerettete Individuen* in Abschnitt 7.1.5).

Im Bereich der **Regeln für das Evakuieren von Individuen** ( `AbstractEvacuationRule`) haben wir die `EvacuateIndividualsRule` implementiert. Sie wird nur aktiv, wenn ein Individuum eine `ExitCell` betritt. Das Individuum wird als „zu entfernen“ markiert, so dass es am Ende des Zeitschrittes aus der Simulation entfernt wird. Dieses Vorgehen hat zur Folge, dass das Individuum die `ExitCell` noch blockiert, bis alle Individuen in diesem Zeitschritt ihre Bewegung gemacht haben. So wird verhindert, dass im gleichen Zeitschritt mehrere Individuen dieselbe `ExitCell` betreten.

### Konkrete Reaktionsregel

Unsere Implementierung der `AbstractReactionRule` ist die `PersonReactRule`. Diese sorgt dafür, dass Personen nach Ablauf ihrer Reaktionszeit (ihrer *reactiontime*, siehe Abschnitt 6.3.3) alarmiert werden und sich auf den Weg zum Ausgang machen. Dazu wird die verbleibende Reaktionszeit eines Individuums in jedem Zeitschritt heruntergezählt. Spätestens, wenn die Reaktionszeit verstrichen ist, wird die Person dann alarmiert.

Außerdem haben wir uns überlegt, dass Personen, die sich im gleichen Raum aufhalten, einander gegenseitig alarmieren. Damit wollen wir das Phänomen verhindern, dass die Personen aus einem Raum erst nach und nach loslaufen, da uns dies unrealistisch erscheint. Wenn die Reaktionszeit einer Person in einem Raum abgelaufen ist, werden deshalb automatisch auch alle anderen Personen, die sich in diesem Raum befinden, alarmiert, auch wenn ihre Reaktionszeit noch nicht abgelaufen ist. Zusätzlich werden alle Personen in einem Raum alarmiert (sofern dies noch nicht geschehen ist), wenn eine alarmierte Person den Raum betritt.

### Reihenfolge der Regeln

Im `DefaultRuleSet` werden die Regeln in dieser Reihenfolge ausgeführt:

**Initialisierungs-Regeln:**

- `InitialPotentialConcreteRule`
- `SaveIndividualsRule`
- `EvacuateIndividualsRule`
- `InitialDynamicPotentialRule`

**Schleifen-Regeln:**

- `PersonReactRule`
- `WaitingMovementRule`
- `SaveIndividualsRule`
- `EvacuateIndividualsRule`
- `ChangePotentialFamiliarityOrAttractivityRule`
- `ChangePotentialInsufficientAdvancementRule`

### 7.1.7. Parameter und Methoden des DefaultParameterSets

Grundsätzlich unterscheiden wir zwei Arten von Parametern: Zum einen gibt es in der Simulation **Attribute**, die jedes Individuum hat, zum anderen gibt es **Gewichtungsparameter**, die u.A. den Einfluss dieser Attribute auf die Simulation beschreiben. Die Individuenattribute werden in der Klasse `Individual`, die Gewichtungsparameter in der Klasse `ParameterSet` gespeichert. Dort werden auch Methoden zur Aktualisierung von Individuenattributen oder zur Berechnung von anderen Werten festgelegt (wobei diese Aktualisierungen/Berechnungen von Parametern abhängig sind).

#### Individuenattribute

Die Individuenattribute und ihre Initialisierungen wurden in Kapitel 6.3.3 erklärt. Es folgt eine Übersicht über ihre Bedeutung und Verwendung innerhalb des Zellulären Automaten mit dem `DefaultParameterSet` und dem `DefaultRuleSet`. Hier wird auch beschrieben, ob die Attribute dynamisch oder statisch sind, d.h. ob ein Attribut während der Simulation verändert wird oder nicht.

**age** ist das Alter (statisch). Es wird nur bei der Umwandlung der Individuenattribute benötigt und bestimmt dabei die Werte von **exhaustionFactor**, **maxSpeed** und **reactiontime** (in den Methoden `getExhaustionFromAge`, `getSpeedFromAge` und `getReactiontimeFromAge`).

**familiarity** ist die Ortskenntnis (statisch). Sie hat Einfluss auf die Anzahl an Ausgängen, die einem Individuum bekannt sind, und wird daher in den Potentialregeln verwendet (siehe Abschnitt *Konkrete Potentialregeln* in Kapitel 7.1.6).

**panic** ist der aktuelle Panikwert (dynamisch). Er beeinflusst, ob das Potential gewechselt wird (Methode `changePotentialThreshold`), wie sehr sich ein Individuum an sein statisches Potential hält (Methode `effectivePotential`) und wie schnell das Individuum laufen möchte (Methode `updateSpeed`). Der Panikwert wird in der Methode `updatePanic` aktualisiert.

**panicFactor** gibt an, wie langsam oder schnell jemand in Panik verfällt (statisch). Der `panicFactor` beeinflusst in `updatePanic` die Aktualisierung des Panikwertes.

**slackness** gibt an, in welchem Maße jemand trödelt (statisch). Anhand der `slackness` wird in `idleThreshold` die Wahrscheinlichkeit dafür bestimmt, ob ein Individuum trödelt.

**exhaustion** ist die aktuelle Erschöpfung (dynamisch). Sie beeinflusst in `updateSpeed`, wie schnell das Individuum laufen möchte. Die Erschöpfung wird in der Methode `updateExhaustion` aktualisiert.

**exhaustionFactor** gibt an, wie langsam oder schnell ein Individuum erschöpft (statisch). Er beeinflusst in `updateExhaustion` die Aktualisierung der Erschöpfung.

**currentSpeed** ist die aktuelle Wunschgeschwindigkeit (dynamisch). Sie wird in `updateExhaustion` benötigt, in den Bewegungsregeln zur Berechnung der tatsächlichen Geschwindigkeit benutzt (dort wird sie noch von Verzögerungsfaktoren von Treppen oder Verzögerungsbereichen beeinflusst) und in `updateSpeed` aktualisiert.

**maxSpeed** ist die Maximalgeschwindigkeit des Individuums (statisch). Sie beeinflusst in `updateExhaustion` bzw. `updateSpeed` die Aktualisierungen von Erschöpfung bzw. Wunschgeschwindigkeit.

**reactiontime** ist die Reaktionszeit (dynamisch). Sie wird in der Reaktionsregel heruntergezählt und bestimmt dort mit, ob ein Individuum alarmiert ist und somit in den Bewegungsregeln auch wirklich bewegt wird (siehe auch *Konkrete Reaktionsregeln* im Abschnitt 7.1.6).

### Gewichtungsparameter

Die Werte der Gewichtungsparameter können in unserem Tool ZET zur Laufzeit geändert werden. Inwieweit sie in die Simulation eingehen, hängt von dem verwendeten Regelsatz und Parametersatz ab.
Wir beschreiben nun die Verwendung der einzelnen Gewichtungsparameter mit dem `DefaultRuleSet` und dem `DefaultParameterSet`.

**Gewichtung des dynamischen Potentials** Der Gewichtungsparameter **dynamicPotentialWeight** wird in der Methode `effectivePotential` benutzt und gibt den Einfluss des dynamischen Potentials an.

**Gewichtung des statischen Potentials** Der Gewichtungsparameter **staticPotentialWeight** wird in der Methode `effectivePotential` benutzt und gibt den Einfluss des statischen Potentials an.

**Einfluss der Panik auf Potentiale** Der Gewichtungsparameter **panicWeightOnPotentials** wird in der Methode `effectivePotential` benutzt und gibt den Einfluss des Parameters **panic** auf die Gewichtung zwischen dynamischem und statischem Potential an.

**Ausbreitungswahrscheinlichkeit des dynamischen Potentials** Der Gewichtungsparameter **probabilityDynamicIncrease** wird im `PotentialController` benutzt, um zu bestimmen, ob das dynamische Potential sich ausbreitet (Genaueres siehe Abschnitt *Dynamisches Potential* in 7.1.4).

**Verfallswahrscheinlichkeit des dynamischen Potentials** Der Gewichtungsparameter **probabilityDynamicDecrease** wird im `PotentialController` benutzt, um zu bestimmen, ob das dynamische Potential sinkt (Genaueres siehe Abschnitt *Dynamisches Potential* in 7.1.4).

**Ortskenntnis zu Attraktivität** Der Gewichtungsparameter **probability-**`ChangePotentialFamiliarityOrAttractivityOfExitRule` wird in der `ChangePotentialFamiliarityOrAttractivityOfExitRule` als Wahrscheinlichkeit dafür benutzt, dass die `ChangePotentialFamiliarityRule` angewendet wird (oder andernfalls die `ChangePotentialAttractivityOfExitRule`, siehe auch Abschnitt *Konkrete Potentialregeln* in 7.1.6).

**Globale maximale Geschwindigkeit** Der Gewichtungsparameter **AbsoluteMaxSpeed** legt den Wert der größten Geschwindigkeit, die ein Individuum in der Simulation überhaupt laufen kann, fest und wird in $\frac{m}{sec}$ angegeben. Die Maximal- und Wunschgeschwindigkeiten der Individuen werden relativ zur absoluten maximalen Geschwindigkeit als Faktor aus $[0, 1]$ angegeben. So verändert sich der Einfluss anderer Werte auf die Geschwindigkeit nicht, wenn die absolute maximale Geschwindigkeit verändert wird. Die Berechnung der Maximalge-

schwindigkeiten der Individuen wurde bereits in Abschnitt 6.3.3 beschrieben. Dort wird die Methode `getSpeedFromAge` benutzt. Um korrekte Werte für die Maximalgeschwindigkeiten zu erhalten, geht in diese Methode die `AbsoluteMaxSpeed` ein.

Außerdem wird die globale Maximalgeschwindigkeit auch dafür benutzt, die Zeitschritte des Zellulären Automaten in Sekunden umzurechnen: Ein Zeitschritt dauert $\frac{0{,}4}{\textbf{AbsoluteMaxSpeed}}$ Sekunden.

**Potentialwechsel im Panikfall** Der Gewichtungsparameter **panicToProbOfPotentialChangeRatio** wird in der Methode `changePotentialThreshold` benutzt und beeinflusst dort die Wahrscheinlichkeit, dass das Potential gewechselt wird.

**Trödel-Übersetzungs-Faktor** Der Gewichtungsparameter **slacknessToIdleRatio** wird in der Methode `idleThreshold` benutzt, um für die Berechnung der Wahrscheinlichkeit für das Trödeln das Individueattribut **slackness** zu skalieren.

**Panikverringerung** Der Gewichtungsparameter **panicDecrease** wird in der Methode `updatePanic` benutzt, um den Panikwert des Individuums zu verringern, falls das Individuum eine Zelle, die es mit hoher Priorität ausgewählt hat, erreicht.

**Panikerhöhung** Der Gewichtungsparameter **panicIncrease** wird in der Methode `updatePanic` benutzt, um den Panikwert des Individuums zu erhöhen, falls das Individuum eine Zelle, die es mit sehr niedriger Priorität ausgewählt hat, erreicht.

**Panikschwellenwert** Der Gewichtungsparameter **panicThreshold** wird in der Methode `updatePanic` benutzt, in der er festlegt, wann die Panik eines Individuums steigt oder sinkt.

**Panikeinfluss auf Geschwindigkeit** Der Gewichtungsparameter **panicWeightOnSpeed** wird in der Methode `updateSpeed` benutzt und beinflusst die neue Wunschgeschwindigkeit des Individuums. Er gewichtet den Einfluss des Parameters **panic**, der dafür sorgt, dass das Individuum schneller laufen möchte.

**Erschöpfungseinfluss auf Geschwindigkeit** Der Gewichtungsparameter **exhaustionWeightOnSpeed** wird in der Methode `updateSpeed` benutzt

und beinflusst die neue Wunschgeschwindigkeit des Individuums. Er gewichtet den Einfluss des Parameters **exhaustion**, der dafür sorgt, dass das Individuum langsamer laufen möchte. Der Einfluss der Erschöpfung auf die Geschwindigkeit ist also dem Einfluss der Panik entgegengesetzt.

### Methoden des `DefaultParameterSet`

Wir beschreiben nun die Methoden unseres `DefaultParameterSets`, das in unserer Software als Standard eingestellt ist. Dabei unterscheiden wir nach Methoden, die Individuenattribute verändern und weiteren Methoden.

**Aktualisierungen von Individuenattributen** Das `DefaultParameterSet` enthält die folgenden Methoden zur Aktualisierung von Individuenattributen:

**updatePanic** Diese Methode wird von den Bewegungsregeln aufgerufen, damit der Panikwert geändert wird, falls sich ein Individuum bewegt oder bewegen möchte, aber blockiert wird. Die Regel übergibt dabei das Individuum, die ausgewählte Zielzelle und eine Liste von Wunschzielzellen des Individuums, die nach Prioritäten (aufsteigend) sortiert ist. Die erste Priorität hat dabei die Zelle, auf die das Individuum am liebsten gehen würde; die nächsthöhere Priorität hat die Zelle, auf die es am zweitliebsten gehen würde usw. Aus diesen Zellen wurde in der Bewegungsregel nun schon eine ausgesucht, die wirklich frei ist und auf die das Individuum nun gehen wird: dies ist die Zielzelle.

Mit Hilfe der Attribute **panic** und **panicFactor** des Individuums, den Gewichtungsparametern **panicIncrease**, **panicDecrease** und **panicThreshold**, der Zielzelle und der Wunschzielzellenliste wird mit dieser Methode nun der neue Wert für das Attribut **panic** berechnet.

Die Entscheidung, ob die Panik dabei erhöht oder gesenkt wird, hängt davon ab, die wie vielte Zelle die Zielzelle in der nach Priorität sortierten Wunschliste ist. Die Zellen mit höherer Priotität wären eigentlich besser gewesen als die Zelle, auf die das Individuum geht. Wir nennen diese Zellen übersprungene Zellen.

Ist die Anzahl übersprungener Zellen zu groß, steigt die Panik. Ist sie hingegen klein, dann sinkt die Panik. Der Schwellenwert, ab dem die Anzahl der übersprungenen Zellen zu einer Panikerhöhung führt, wird aus dem Gewichtungsparameter **panicThreshold** ausgelesen. Bei der Veränderung der Panik gilt: Je weniger Zellen übersprungen werden, desto stärker verringert sich die Panik; je mehr Zellen übersprungen werden, desto stärker erhöht sich die Panik.

Die genaue Berechnung nutzt die Gewichtsparameter **panicIncrease** und **panicDecrease**, welche die Werte angeben, um die sich die Panik erhöht bzw. verringert. Diese können zusätzlich noch durch den individuenabhängigen Parameter **panicFactor** für jedes Individuum skaliert werden.

Falls weniger als **panicThreshold** Zellen übersprungen werden, wird die Panik mit der Formel

$$\mathrm{n} = \mathrm{p} - \mathrm{panicFactor} * \mathrm{panicDecrease} * (\mathrm{panicThreshold} - \mathrm{failures}).$$

verringert, andernfalls wird sie mit der Formel

$$\mathrm{n} = \mathrm{p} + \mathrm{panicFactor} * \mathrm{panicIncrease} * (\mathrm{failures} - \mathrm{panicThreshold})$$

erhöht. Dabei ist $n$ der neue Panikwert, $p$ der alte Panikwert und `failures` bezeichnet die Anzahl der übersprungenen Zellen.

**updateExhaustion** Auch diese Methode wird von den Bewegungsregeln aufgerufen. Übergeben werden das Individuum und die Zielzelle.
Aus den Attributen **exhaustion**, **exhaustionFactor**, **currentSpeed** und **maxSpeed** des Individuums und der Zielzelle wird der neue Wert für **exhaustion** berechnet.
Dabei sinkt die Erschöpfung, wenn der Anteil der Wunschgeschwindigkeit an der Maximalgeschwindigkeit weniger als 50% beträgt und steigt, wenn der Anteil mehr als 50% beträgt. Bei genau 50% verändert sich die Erschöpfung nicht. Der Wert, um den die Erschöpfung sinkt oder steigt, beträgt dabei jeweils die Differenz des Anteils zu 50%, die noch mit dem **exhaustionFactor** gewichtet wird (Da die Wunschgeschwindigkeit nicht anzeigt, wenn ein Individuum in einem Schritt stehenbleibt, muss dieser

Fall gesondert behandelt werden. Er wird daran erkannt, dass die Zelle des Individuums gleich der übergebenen Zielzelle ist. In diesem Fall wird statt **currentSpeed** der Wert 0 für die Berechnung benutzt).
Genaue Berechnung:

$$\text{newExhaustion} = \text{exhaustion} + \left(\frac{\text{currentSpeed}}{\text{maxSpeed}} - 0.5\right) \cdot \text{eFactor}$$

Hier wurde *exhaustionFactor* durch *eFactor* abgekürzt. Falls *newExhaustion* außerhalb des Intervalls $[0, 0.99]$ liegt, wird die entsprechende Grenze des Intervalls als neuer Wert für **exhaustion** verwendet, ansonsten der Wert von *newExhaustion*.

**updateSpeed** Auch `updateSpeed` wird von den Bewegungsregeln aufgerufen, benötigt aber nur das Individuum.
Aus den Attributen **panic**, **exhaustion** und **maxSpeed** des Individuums, und den Gewichtungsparametern **panicWeightOnSpeed** und **exhaustionWeightOnSpeed** wird die neue Wunschgeschwindigkeit des Individuums berechnet.
Die Idee dabei ist, dass das Individuum um den gewichteten Panikwert schneller als seine Maximalgeschwindigkeit laufen möchte und um den gewichteten Erschöpfungswert langsamer.
Genaue Berechnung:

$$\begin{aligned}\text{newSpeed} = {} & \text{maxSpeed} + (\text{panic} \cdot \text{panicWeightOnSpeed}) \\ & - (\text{exhaustion} \cdot \text{exhaustionWeightOnSpeed})\end{aligned}$$

Falls *newSpeed* außerhalb des Intervalls $[0.0001, \textbf{maxSpeed}]$ liegt, wird die entsprechende Grenze des Intervalls als neuer Wert für **currentSpeed** genommen, ansonsten der Wert von *newSpeed*.

**Weitere Berechnungen** Zum Abschluss dieses Abschnitts beschreiben wir noch einige weitere wichtige Methoden, die von den Regeln des `DefaultParameterSets` benutzt werden.

**idleThreshold** Diese Methode wird von den Bewegungsregeln benutzt, um zu bestimmen, ob das Individuum trödelt.

Das Attribut **slackness** des übergebenen Individuums wird mit dem Gewichtungsparameter **slacknessToIdleRatio** multipliziert und zurückgegeben. Die Bewegungsregeln benutzen diesen Wert als Wahrscheinlichkeit dafür, dass das Individuum trödelt.

**changePotentialThreshold** Diese Methode wird von den Regeln `ChangePotentialAttractivityOfExitRule`, `ChangePotentialFamiliarityRule` und `ChangePotentialFamiliarityOrAttractivityOfExitRule` benutzt, um festzustellen, ob diese Regel für das übergebene Individuum angewendet werden soll.
Das Produkt des Attributs **panic** des Individuums und des Gewichtungsparameters **panicToProbOfPotentialChangeRatio** wird zurückgegeben und von den Regeln als Wahrscheinlichkeit dafür benutzt, ob sie angewendet werden. (Im Sonderfall, dass das Individuum schon gerettet ist, wird 0 zurückgegeben, so dass so ein Individuum sein Potential nicht mehr wechselt.)

**effectivePotential** Diese Methode wird in den Bewegungsregeln dazu benutzt, Wahrscheinlichkeiten für die Wahl der Zielzelle zu berechnen und die Wunschzielzellenliste des Individuums aufzustellen, die für `updatePanic` benötigt wird.
Es werden zwei Zellen übergeben, die erste ist die Zelle, auf der das Individuum steht, die zweite sollte eine der Nachbarzellen dieser Zelle sein. Die Methode rechnet eine effektive Potentialwertdifferenz zwischen diesen zwei Zellen aus. Diese Differenz wird von den Potentialwerten der beiden Zellen sowohl bezüglich des statischen als auch bezüglich des dynamischen Potentials des Individuums, das auf der ersten Zelle steht, beeinflusst. Weiterhin fließen das Attribut **panic** des Individuums, und die Gewichtungsparameter **dynamicPotentialWeight**, **staticPotentialWeight** und **panicWeightOnPotentials** in die Berechnung mit ein. Die Entscheidung zwischen – bzw. die „Mischung“ aus – dynamischem und statischem Potential simuliert die Entscheidung dazwischen, ob das Individuum mehr dem „Herdentrieb“ folgt (dynamisches Potential) oder eher seinem eigenen Weg folgt (statisches Potential). Wir berechnen das effektive Potential so, dass der Einfluss des dynamischen Potential mit höherer Panik steigt und entsprechend bei kleinerer Panik sinkt.

Das Ergebnis der Berechnung

$$\begin{aligned} &\text{panic}^{\text{panicWeightOnPotentials}} \cdot \text{dynamicPotentialWeight} \cdot \text{dynPotDiff} \\ &+ (1 - \text{panic})^{\text{panicWeightOnPotentials}} \cdot \text{staticPotentialWeight} \cdot \text{statPotDiff} \end{aligned}$$

wird zurückgegeben. Dabei ist *dynPotDiff* die Differenz der Potentialwerte der beiden Zellen, die sie bezüglich des dynamischen Potentials haben, und *statPotDiff* dieselbe Differenz bezüglich des statischen Potentials.

**Methoden für die Umwandlungen der Indivdiuenparameter** Die Methoden `getExhaustionFromAge`, `getReactiontimeFromAge`, `getSlacknessFromDecisiveness` und `getSpeedFromAge` wurden bereits in Kapitel 6.3.3 beschrieben.

## 7.2. Graphalgorithmen

Um Evakuierungsprobleme mit Hilfe von Flussalgorithmen angehen zu können, haben wir neben den bereits erläuterten Datenstrukturen für Graphen zunächst grundlegende Graphalgorithmen implementiert, wie unter anderem:

- Breitensuche in Graphen
- Dijkstras Algorithmus
- Algorithmus von Moore, Bellman und Ford
- Tiefensuche in Graphen
- Topologische Sortierung von Graphen

Aufbauend auf den Datenstrukturen für Graphen und Funktionen sowie diesen grundlegenden Algorithmen haben wir dann klassische Flussalgorithmen als Grundlage für die komplexeren Flussalgorithmen mit zeitlicher Komponente implementiert. Dabei haben wir verschiedene Algorithmen für die folgenden drei klassischen (statischen) Flussprobleme umgesetzt:

- Maximum Flow Problem
- Minimum Cost Flow Problem

- Transshipment Problem

Für die Lösung von Evakuierungsproblemen haben wir vier Flussprobleme mit einer zeitlichen Komponente in Betracht gezogen und für alle Algorithmen implementiert, in der Regel durch Reduktion auf eines der oben genannten Probleme. Die vier von uns unterstützten (dynamischen) Flussprobleme sind:

- Maximum Flow Over Time Problem
- Transshipment Over Time Problem
- Quickest Transshipment Problem
- Earliest Arrival Transshipment Problem

Wie auch in den entsprechenden Abschnitten ausgeführt, hat sich das Earliest Arrival Transshipment Problem als das am besten geeignete Problem herausgestellt, um Evakuierungsprobleme mittels Flussalgorithmen zu modellieren. Zum einen, weil dieses Problem Evakuierungsprobleme am genausten abbildet, zum anderen, weil in der Praxis schnelle Algorithmen dafür verfügbar sind.

Im Folgenden gehen wir auf die von uns betrachteten Probleme und die Algorithmen, die wir zur ihrer Lösung implementiert haben, ein.

### 7.2.1. Maximum Flow Problem

Das **Maximum Flow Problem** ist eines der grundlegendsten Flussprobleme. Im Kontext von Evakuierungsproblemen können wir es wie folgt definieren: Gegeben ist ein Netzwerk $G = (V, E)$ mit Kapazitäten $c : E \to \mathbb{N}_0$ sowie Quellen $S^+ \subseteq V$ und Senken $S^- \subseteq V$ mit $S^+ \cap S^- = \emptyset$. Gesucht ist ein maximaler zulässiger Fluss.

Auch wenn das **Maximum Flow Problem** in unserem Kontext keine direkte Anwendung haben mag, so lassen sich Probleme, die eine direkte Anwendung haben, darauf reduzieren. Dies geschieht teilweise unter der Verwendung von zeitexpandierten Netzwerken, was bedeutet, dass potentiell **Maximum Flow Probleme** auf sehr großen Graphen gelöst werden müssen. Somit ist es essentiell, einen effizienten Algorithmus zur Lösung dieses Problems zur Verfügung zu haben.

Basierend auf den Resultaten von Cherkassy und Goldberg [8] haben wir daher den **Preflow Push** Algorithmus mit der **Highest Label** Strategie implementiert. Zusätzlich unterstützen wir die in obigem Paper beschriebenen **Global-** und **Gap**-Heuristiken. Die Worst-Case Laufzeit dieses Ansatzes ist $O(n^2\sqrt{m})$, wobei $n := |V|$ und $m := |E|$.

### 7.2.2. Transshipment Problem

Im Falle des **Transshipment Problems** ist ein Netzwerk $G = (V, E)$ mit Kapazitäten $c : E \to \mathbb{N}_0$ und Angeboten / Bedarfen $b : V \to \mathbb{Z}$ gegeben. Gesucht ist ein zulässiger Fluss, der alle Angebote und Bedarfe erfüllt, falls ein solcher existiert.

Dieses Problem lässt sich durch Reduktion auf das **Maximum Flow Problem** lösen, die Worst-Case Laufzeit beträgt somit auch hier $O(n^2\sqrt{m})$. Verwendung findet auch dieses Problem nicht direkt, sondern als Grundlage für das **Transshipment Over Time Problem**, das weiter unten näher ausgeführt wird.

### 7.2.3. Minimum Cost Flow Problem

Das **Minimum Cost Flow Problem** ist gegeben durch ein Netzwerk $G = (V, E)$ mit Kapazitäten $c : E \to \mathbb{N}_0$, Kosten $t : E \to \mathbb{N}_0$ und Angeboten / Bedarfen $b : V \to \mathbb{Z}$. Gesucht ist ein zulässiger Fluss mit minimalen Kosten, der alle Angebote und Bedarfe erfüllt.

Diese Problemstellung ist für uns im Rahmen von Evakuierungen interessant, da wir die Fahrzeiten (bzw. Laufzeiten) in unseren Gebäuden als Kosten betrachten können und diese Fahrzeiten möglichst gering halten wollen. Wir haben daher zwei verschiedene Lösungsstragetien für diese Problemstellung umgesetzt.

Zum einen haben wir den **Minimum Mean Cycle Cancelling**-Algorithmus nach Korte & Vygen [23] implementiert. Dieser Algorithmus bestimmt mit Hilfe des oben beschriebenen **Transshipment**-Problems eine zulässige Lösung (d.h. einen Fluss, der alle Angebote und Bedarfe erfüllt) und verbessert danach schrittweise die Kosten der Lösung. Jeder Verbesserungsschritt wird mit Hilfe von dynamischer Programmierung durchgeführt und erfordert $O(nm)$ Zeit. Die

Anzahl der Verbesserungsschritte ist maximal $O(mn \log(n(|t_{min}|+1)))$ , wobei $t_{min}$ die minimalen Kosten einer Kante sind. Alternativ lässt sich die Anzahl der Schritte durch $O(m^2 n \log n)$ beschränken (vgl. Tardos [31]), allerdings ist für unsere Netzwerke die obere Schranke schärfer, da $|t_{min}|$ in der Regel 0 oder 1 ist.

Zum anderen haben wir, basierend auf Korte & Vygen [24], einen **Successive Shortest Path**-Algorithmus implementiert. Dieser beginnt mit einem Fluss mit minimalen Kosten, der in der Regel die Angebote und Bedarfe nicht erfüllt und stellt schrittweise die Erfüllung dieser Angebote und Bedarfe her. Jeder Schritt erfordert $O(nm)$ Zeit und besteht im wesentlichen aus einem Aufruf des Algorithmus von **Moore-Bellman-Ford**. Es sind maximal $B$ Schritte notwendig, wobei $B$ die Summe aller Angebote ist. Damit ist die Worst-Case-Laufzeit dieses Ansatzes $O(Bnm)$. Da $B$ im Vergleich zu $n$, $m$ in unseren Netzwerken typischerweise klein ist, ist dieser Ansatz für unsere Evakuierungsprobleme besser geeignet als der vorige. **Capacity Scaling**, wie von Edmonds und Karp [11] beschrieben, nutzen wir nicht, da unsere Netzwerke nur in den seltesten Fällen Pfade mit großen Kapazitäten besitzen.

### 7.2.4. Maximum Flow Over Time Problem

Im **Maximum Flow Over Time Problem** ist ein Netzwerk $G = (V, E)$ mit Quellen $S^+ \subseteq V$, Senken $S^- \subseteq V$, Kapazitäten $c : E \to \mathbb{N}_0$, Fahrzeiten $t : E \to \mathbb{N}_0$ und einem Zeithorizont $T$ gegeben. Gesucht ist ein zulässiger dynamischer Fluss, der innerhalb des Zeithorizontes möglichst viele Flusseinheiten von den Quellen $S^+$ zu den Senken $S^-$ transportiert.

Das **Maximum Flow Over Time Problem** ist eines der einfachsten Flussprobleme mit zeitlicher Komponente. Dadurch ist es zwar auch einfacher zu lösen, andererseits aber für die Modellierung von Evakuierungsproblemen nur begrenzt geeignet.

Wir unterstützen die Lösung des Problems auf zweierlei Weise. Einerseits durch Reduktion auf ein **Minimum Cost Flow Problem** und einer damit verbundenen Worst-Case Laufzeit von $O(m^2 n^2 \log(n|t_{min}|)$, andererseits durch Reduktion auf ein **Maximum Flow Problem** im zeitexpandierten Netzwerk,

was eine Worst-Case-Laufzeit von $O(n^2T^2\sqrt{mT})$ zur Folge hat. Welcher Ansatz schneller ist, hängt dabei sehr von dem betrachteten Problem ab.

### 7.2.5. Transshipment Over Time Problem

Das **Transshipment Over Time Problem** ist gegeben durch ein Netzwerk $G = (V, E)$ mit Kapazitäten $c : E \to \mathbb{N}_0$, Fahrzeiten $t : E \to \mathbb{N}_0$, Angeboten / Bedarfen $b : V \to \mathbb{Z}$ und einem Zeithorizont $T$. Gesucht ist ein zulässiger dynamischer Fluss, der alle Angebote und Bedarfe innerhalb des Zeithorizonts erfüllt.

Diese Problemstellung ist für Evakuierungsprobleme äquivalent zu der Frage, ob ein Gebäude innerhalb einer bestimmten Zeit evakuiert werden kann, wenn sich die Menschen „intelligent" verhalten. Wir lösen das **Transshipment Over Time Problem** durch Reduktion auf ein **Transshipment Problem**, was eine Worst-Case Laufzeit von $O(n^2T^2\sqrt{mT})$ bedeutet.

### 7.2.6. Quickest Transshipment Problem

Das **Quickest Transshipment Problem** besteht aus einem Netzwerk $G = (V, E)$ mit Kapazitäten $c : E \to \mathbb{N}_0$, Fahrzeiten $t : E \to \mathbb{N}_0$ und Angeboten / Bedarfen $b : V \to \mathbb{Z}$. Gesucht ist die minimale Zeit, die nötig ist, um alle Angebote und Bedarfe zu erfüllen.

Im Kontext von Evakuierungsproblemen entspricht dies der Bestimmung einer für eine Evakuierung nötigen Zeit, unter der Annahme intelligenten Verhaltens der Beteiligten. Dieses Problem lässt sich durch geometrische Suche des optimalen Zeithorizontes mittels Reduktion auf das **Transshipment Over Time Problem** lösen. Somit ist die Worst-Case Laufzeit $O(n^2T^2\sqrt{mT}\log T)$.

### 7.2.7. Earliest Arrival Transshipment Problem

Das **Earliest Arrival Transshipment Problem** besteht aus einem Netzwerk $G = (V, E)$ mit Kapazitäten $c : E \to \mathbb{N}_0$, Fahrzeiten $t : E \to \mathbb{N}_0$, Angeboten / Bedarfen $b : V \to \mathbb{Z}$ und einem Zeithorizont $T$. Wir nehmen an, dass das

Problem nur eine einzige Senke besitzt, damit die Existenz eines **Earliest Arrival Transshipments** gesichert ist. Gesucht ist zulässigerdynamischer Fluss, der alle Angebote und Bedarfe erfüllt und für jeden Zeitpunkt die maximale Flussmenge in die Senke schickt.

Diese Problemstellung ist für uns die mit Abstand interessanteste, da wir im Fall einer Evakuierung möglichst schnell möglichst viele Menschen evakuieren wollen. Die Einschränkung auf eine Senke ist für uns unproblematisch, da alle sicheren Bereiche im Netzwerk durch eine Senke zusammengefasst werden können. Wir unterstützen mehrere Ansätze zur Lösung dieser Problemstellung.

Einerseits durch Reduktion auf die oben aufgeführten Probleme, in dem zunächst mittels eines **Quickest Transshipment Problems** der optimale Zeithorizont bestimmt wird und anschließend mittels eines **Minimum Cost Flow Problems** im zeitexpandierten Netzwerk ein **Earliest Arrival Transshipment** bestimmt wird. Durch Kombination mit dem **Successive Shortest Path** Algorithmus ergibt sich also eine Worst-Case Laufzeit von $O(BnmT^2)$.

Andererseits unterstützen wir eine „direkte“ Berechnung eines **Earliest Arrival Transshipment** durch den **Successive Earliest Arrival Augmenting Path** Algorithmus von Tjandra [32]. Dieser benutzt kein explizit konstruiertes zeitexpandiertes Netzwerk, was diesen Algorithmus in der Praxis sehr schnell macht. Die Worst-Case Laufzeit des Algorithmus ist ebenfalls $O(BnmT^2)$. Aufgrund der guten praktischen Eigenschaften empfehlen wir die Benutzung von Tjandras Algorithmus für Evakuierungsprobleme.

## 7.3. Fluchtpläne

Im folgenden Abschnitt möchten wir beschreiben, auf welche Weise wir die Nutzung von Simulationen und Flussalgorithmen kombinieren. Dabei werden auch neue Datenstrukturen eingeführt, so dass die Beschreibung eigentlich teilweise besser in Kapitel 5 passen würde. Da es sich aber um einen abgeschlossenen Teil des Programms handelt und die Datenstrukturen mehr als in den anderen Teilen direkt aus der umgesetzten Idee entstanden sind, möchten wir die Erklärung nicht künstlich aufteilen.

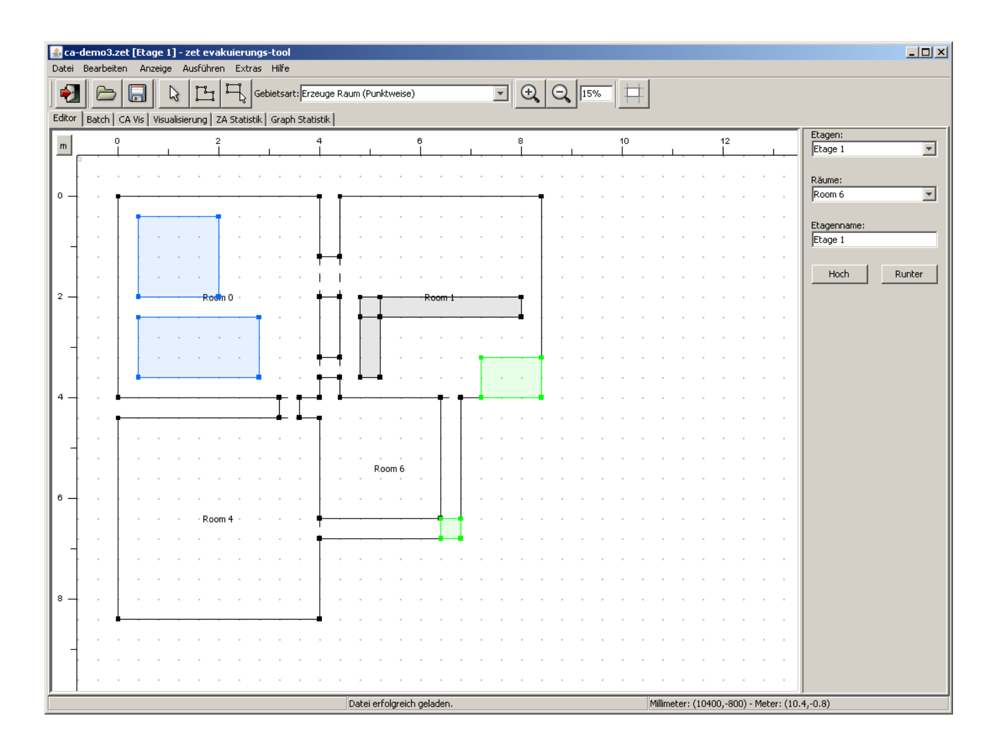



**Abbildung 7.1.:** Grundriss des in Abschnitt 7.3.4 betrachteten Beispiels.

Wir verfolgen bei unseren Fluchtplansimulationen die Frage, ob Evakuierungen tatsächlich reibungslos ablaufen, wenn jedes Individuum den für sich in einer global optimalen Lösung vorgesehenen Fluchtplan verfolgt – oder ob auch dann durch Trödeln und Panik der Ablauf zu sehr gestört wird. Daher erhält jedes Individuum einen **persönlichen Fluchtplan** (Wir beschäftigen uns also ausdrücklich *nicht* mit der Frage, wie man einen auf irgendeine Weise optimalen, globalen Fluchtplan für ein gegebenes Gebäude berechnet). Anschließend wird die Simulation so durchgeführt, dass jedes Individuum seinem persönlichen Fluchtplan folgt. Auf diese Weise erhalten wir eine Art **optimierte Simulation**.

Dadurch stellen die Ergebnisse unserer optimierten Simulation auch eine untere Schranke für die tatsächliche Evakuierungszeit dar (sofern man bei randomisierten Ergebnissen von unteren Schranken sprechen kann).

## 7.3.1. Berechnung der persönlichen Fluchtpläne

Für die persönlichen Fluchtpläne berechnen wir zuerst eine global optimale Lösung mit Hilfe eines Algorithmus für das Earliest Arrival Transshipment

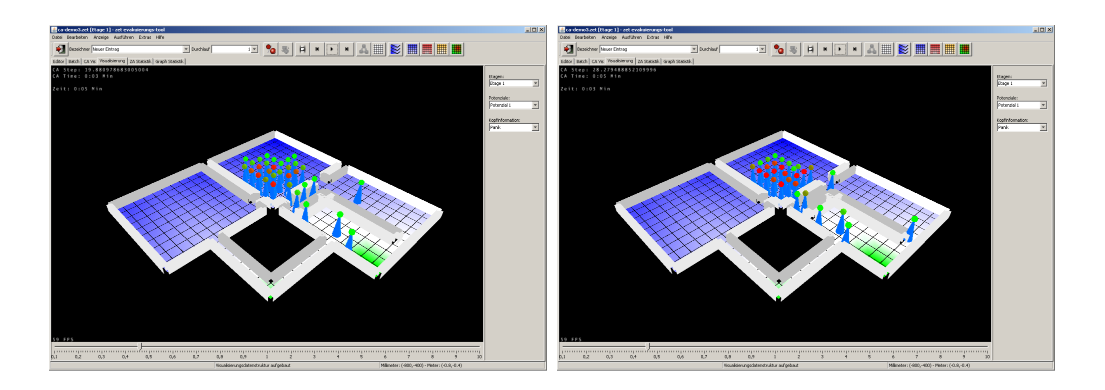


**Abbildung 7.2.:** Simulation des in Abschnitt 7.3.4 behandelten Beispiels *ohne* Verwendung von Optimierung.

Problem (siehe dazu Abschnitt 7.2.5). Dieser liefert uns einen optimalen dynamischen Fluss, der alle Individuen so schnell wie möglich evakuiert und dabei auch zu jedem Zeitpunkt sicherstellt, dass bereits eine bis zu diesem Zeitpunkt maximale Anzahl an Personen evakuiert ist. Außerdem ist der Ergebnisfluss so beschaffen, dass wir ihn in Pfade zerlegen können, über die zu bestimmten Zeitpunkten Individuen „fließen". Wird ein Pfad von mehr als einem Individuum genutzt, duplizieren wir ihn entsprechend oft und erhalten auf diese Weise eine Sammlung von Pfaden für jedes Individuum, die die (global) optimalen Fluchtwege der Individuen angeben. Jedes Individuum erhält also seinen eigenen `DynamicPathFlow`.

Die Knoten unseres Graphen bestehen wie in Abschnitt 6.4.1 beschrieben aus rechteckigen Bereichen des gerasterten Plans. Es ist also möglich, zu jeder Zelle den Knoten anzugeben, in dem diese Zelle liegt. Andersum können wir zu einem Knoten alle Zellen angeben, die von diesem Knoten überdeckt werden. Wir weisen damit die Pfade der (im Fluss ununterscheidbaren) Individuen, die in einem Knoten starten, randomisiert den Individuen zu, die im Zellulären Automaten in diesem Knoten stehen (das sind zumindest gleich viele). Dadurch können wir die berechneten Fluchtpläne tatsächlich auf den Zellulären Automaten übertragen, sofern wir den Bewegungsalgorithmus passend abändern.

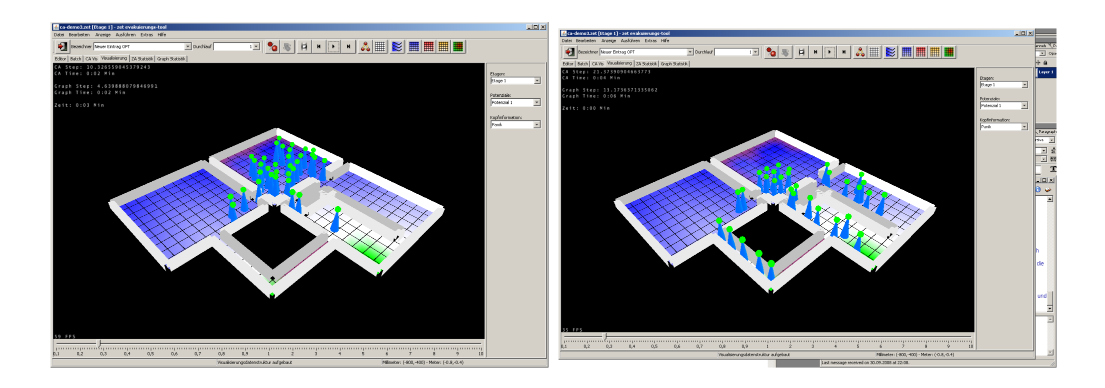


**Abbildung 7.3.:** Simulation des in Abschnitt 7.3.4 behandelten Beispiels *mit* Verwendung von Optimierung.

### 7.3.2. Potentialschläuche

Unsere Individuen bewegen sich während der Simulation des Zellulären Automaten normalerweise anhand eines statischen oder dynamischen Potentials. Stattdessen sollen sie sich nun entsprechend der ihnen zugewiesenen Pfade bewegen. Dabei verlangen wir, dass Individuen niemals von ihrem Pfad abweichen, berücksichtigen aber die zeitlichen Informationen der Earliest Arrival Transsshipment Lösung nicht (diese gibt z.B. auch vor, *wann* ein Individuum beginnt, in Richtung des Ausgangs zu laufen). Letzteres wird schon dadurch erschwert, dass die Individuen in der Flusssimulation ununterscheidbar sind, auch wenn sie im gleichen Knoten starten – verschiedene Individuen im Zellulären Automaten aber sehr wohl verschieden sind, auch, wenn sie sich im gleichen Knoten"quadrat" des Graphen befinden. Dadurch lassen sich Informationen über die Reihenfolge nur randomisiert auf die Individuen übertragen, was wiederum nur zu unnötigen Problemen führt, wenn diese „falschrum" im Knoten stehen.

Damit sich die Individuen an ihren vorgegebenen Pfad halten, definieren wir eine neue Art von Potential, das wir im folgenden **Potentialschlauch** nennen und das in der Klasse `EvacPotential` implementiert ist. Ein Potentialschlauch gehört immer zu dem dem Individuum zugewiesenen `DynamicPathFlow` und verbietet dem Individuum, Zellen zu betreten, die außerhalb der Knoten liegen, die dieser Pfad berührt. Weiterhin wird verhindert, dass das Individuum von Zellen eines Knotens $B$ zu Zellen eines Knotens $A$ geht, sofern $B$ im Pfad hinter $A$ liegt, d.h. Personen dürfen sich auf dem Pfad nur vorwärts bewegen. Inner-

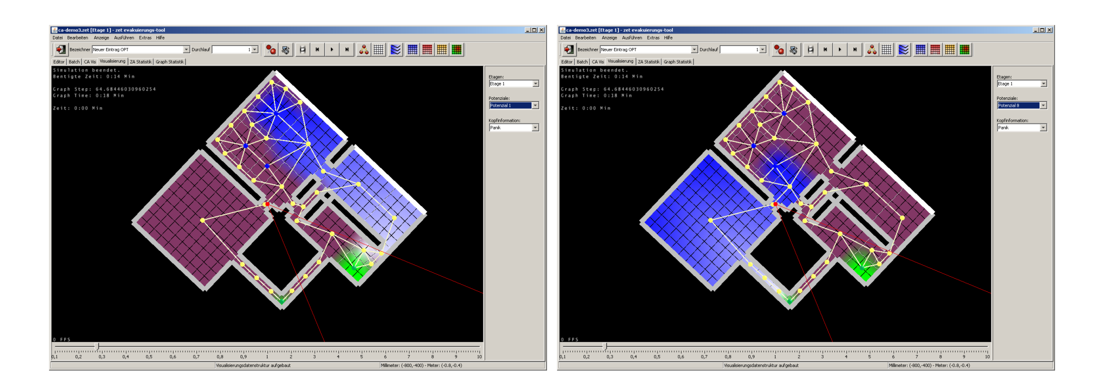


**Abbildung 7.4.:** Zwei der Potentialschläuche, die bei der optimierten Simulation des in Abschnitt 7.3.4 behandelten Beispiels auftreten.

halb der Knoten des `DynamicPathFlows` wird das Potential genauso berechnet wie beim statischen Potential, d.h. abhängig von der Distanz zum gewünschten Ausgang (inklusive Smoothing etc., siehe dazu Abschnitt 7.1.4).

### 7.3.3. Der Quetschregelautomat

Wie bereits erwähnt, bringt die Tatsache, dass die Individuen für die Flussalgorithmen ununterscheidbar sind, im Zellulären automaten aber nicht, einige Schwierigkeiten mit sich. Dies ist leider auch der Fall, wenn wir die zeitliche Komponente von dynamischen Flüssen weitgehend ignorieren: Starten in einem Knoten mehrere Individuen und laufen in verschiedene Richtungen, so kann es zu Staus kommen, wenn wir die Pfade randomisiert so zuweisen, dass sich die Individuen innerhalb des Knotens blockieren (was ja nicht nötig wäre). Besonders problematisch wird dieses, wenn die Knoten so klein sind, dass sich die Individuen tatsächlich blockieren und gar nicht loslaufen.

Zur Lösung dieses Problems haben wir eine zusätzliche Regel implementiert, die es auch in anderen Simulationstools gibt. Der Sinn dieser Regel ist es, dass sich Individuen aneinander „vorbeiquetschen“ können, wenn sie gerne den Platz tauschen möchten. Das Verfahren dazu wird in Abschnitt 7.1.2 beschrieben.

Führt man nun den Quetschregelautomaten mit den passenden `EvacPotentials` aus, so erhält man die gewünschte optimierte Simulation.

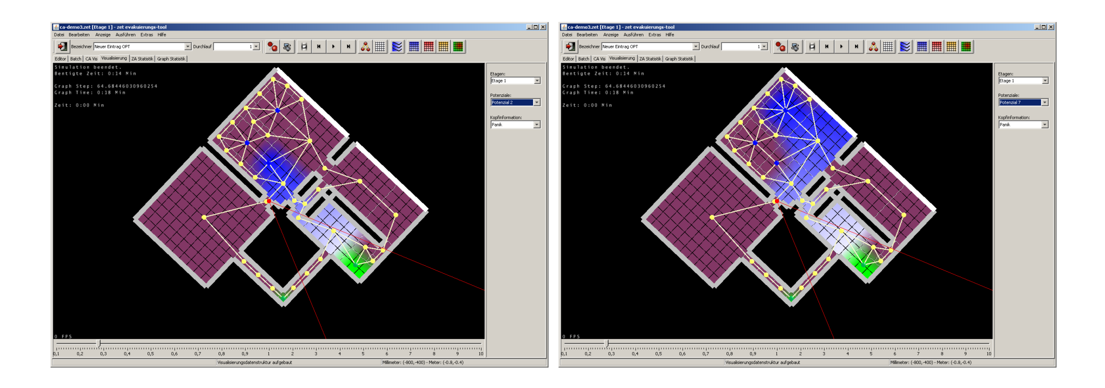


**Abbildung 7.5.:** Zwei weitere Potentialschläuche, die bei der optimierten Simulation des in Abschnitt 7.3.4 behandelten Beispiels auftreten.

### 7.3.4. Erfahrungswerte

Wir haben zum Test unserer optimierten Simulation vor allem Beispiele betrachtet, in denen große Staus entstehen, weil sich alle Personen auf den kürzeren Ausgang stürzen, diesen blockieren und einen anderen Ausgang ignorieren, obwohl dort ein freier Durchgang gewährleistet wäre. Trotz der Behinderungen durch Trödeln und Panik haben sich die Individuen anhand ihrer persönlichen Fluchtpläne so gut verhalten, dass fast die Zeiten der Netzwerkflüsse erreicht wurden. Wir schließen daraus, dass der Einfluss von Eigenschaften wie Trödeln und Panik viel geringer ist als der Einfluss der Ortsunkenntnis und mangelnden Abstimmung der Individuen.

Wir betrachten zum Abschluss noch ein kleines Beispiel. Dazu schauen wir uns das Gebäude an, dessen Grundriss in Abbildung 7.1 dargestellt wird. Alle Individuen befinden sich im größten Raum und sind in zwei Gruppen zusammengefasst. Außerdem gibt es zwei Ausgänge, einen größeren im Nachbarraum und einen etwas versteckten kleineren in einem Verbindungsgang.

Ohne Optimierung versuchen alle Individuen bei der Simulation, den größeren und näher gelegenen Ausgang zu erreichen. Dadurch entsteht schnell ein Stau an der kleinen Tür und die Personen werden panisch. Dieses ist Abbildung 7.2 zu sehen, wobei die Evakuierungsbereiche grün dargestellt werden, der Boden entsprechend des statischen Potentials schattiert ist und die Köpfe der Individuen ihre Panik widerspiegeln (siehe dazu auch das Handbuch in Teil IV).

In der optimierten Simulation hingegen wählen die Individuen beide Ausgänge

und verfolgen dabei auch verschiedene Pfade: Sogar der kleine Nebenraum wird genutzt, um den größeren Ausgang auch von der anderen Seite zu erreichen. Es entsteht fast keine Panik. Zwei Momentaufnahmen aus diesem Ablauf sind in Abbildung 7.3 zu sehen.

Vier der bei der optimierten Simulation benutzten Potentialschläuche haben wir in den Abbildungen 7.4 und 7.5 abgebildet. Die Zellen, die ein Individuum gar nicht betreten darf, sind lila gekennzeichnet. Außerdem ist der verwendete Graph und die entsprechende Aufteilung der Zellen in Knoten eingeblendet.

# Teil III.

# RiMEA-Tests

# Inhaltsverzeichnis

# 8. Validierung/Verifizierung der ZA-Simulationen

Im Folgenden stellen wir die RiMEA-Richtlinie[1] vor. RiMEA ist die Abkürzung für „Richtlinie für Mikroskopische Entfluchtungs-Analysen“ und wurde entworfen, um die Validierung und Verifizierung von Simulationen für Evakuierungen zu unterstützen. Im Anschluss an ausgewählte Textstellen aus der Richtlinie vergleichen wir die gewünschten Ergebnisse mit dem Verhalten unseres Zellulären Automaten.

## 8.1. Die RiMEA-Richtline in Auszügen

„...Zusätzlich zur Einhaltung bauordnungsrechtlicher Anforderungen [...] sind [...] Entfluchtungsberechnungen als Teil eines ganzheitlichen Brandschutzkonzeptes zu empfehlen. ...“, Präambel der RiMEA-Richtlinie.

### 8.1.1. Ziele der RiMEA-Richtline

„...Ziel dieser Richtlinie ist es die Methodik [...] für die Erstellung einer simulationsgestützten Entfluchtungsanalyse festzulegen und:

1. die Gesamtentfluchtungsdauer bzw. die Entfluchtungsdauer von Teilbereichen baulicher Anlagen statistisch zu erfassen und unter Berücksichtigung von sicherheitstechnischen Aspekten zu bewerten;
2. im Einzelfall den Nachweis zu führen, dass die geplanten oder bestehenden Flucht- und Rettungswege abweichend von den Dimensionierungsvorgaben des Bauordnungsrechts für die angenommenen Personenzahlen ausreichen;

3. zu zeigen, dass die Fluchtvorkehrungen ausreichend flexibel sind für den Fall, dass bestimmte Flucht- und Rettungswege oder gesicherte Bereiche aufgrund eines Zwischenfalls nicht verfügbar sind;
4. soweit möglich, signifikante Stauungen die während der Entfluchtung aufgrund der normalen Bewegung von Personen entlang der Flucht- und Rettungswege auftreten zu erkennen.

Diese Richtlinie definiert einen Mindeststandard in Bezug auf die Eingangsgrößen, die Modellbildung, die rechnerische Simulation und die Auswertung und Dokumentation einer Entfluchtungsanalyse. Mit Hilfe der in dieser Richtlinie dargestellten Methodik einer simulationsgestützten Entfluchtungsanalyse soll die Leistungsfähigkeit eines Flucht- und Rettungskonzeptes als Bestandteil einer baulichen Anlage bewertet werden. ...“

Zu diesem Zweck gibt die RiMEA-Richtlinie 13 Tests vor, die von zu validierenden Simulationsprogrammen im Rahmen einiger vorgegebener Parameter durchgeführt werden sollen. Diese und die Ergebnisse *unserer* Entfluchtungssimulationen auf diesen Tests sollen hier vorgestellt werden. Dazu zitieren wir zunächst aus der RiMEA-Richtlinie und geben dann unsere Auswertungen an.

## 8.2. Überprüfung der Komponenten

„Die Überprüfung der Komponenten beinhaltet, zu testen, ob die verschiedenen Komponenten der Software wie vorgesehen funktionieren. Das schließt die Durchführung einer Reihe von elementaren Testfällen ein, um sicherzustellen, dass die wichtigsten Bestandteile des Modells wie beabsichtigt funktionieren. Die folgende Liste ist eine nicht erschöpfende Aufzählung vorgeschlagener Tests, die in den Verifizierungsprozess eingeschlossen werden sollen.“

### 8.2.1. Test 1: Beibehalten der Geschwindigkeit in einem Gang

Bei diesem Test geht es darum, die Geschwindigkeit eines Individuums zu untersuchen, während es einen 40m langen Gang durchquert. Dabei soll überprüft werden, ob die Geschwindigkeit beibehalten wird. In ZET verlangsamt sich

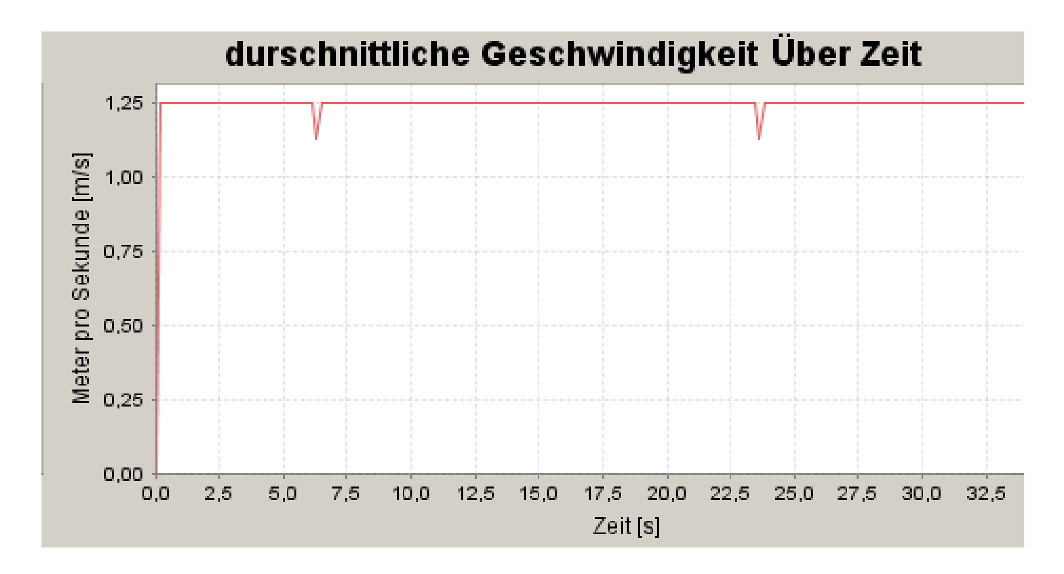



**Abbildung 8.1.:** Durschnittliche Geschwindigkeit über Zeit Rimea-Test 1

normalerweise die Geschwindigkeit der Individuen abhängig von der zurückgelegten Strecke. Dieses Verhalten dient dazu, Erschöpfungszeichen bei den sich schnell bewegenden Personen zu simulieren. Um die gewünschten Bedingungen zu schaffen, wurde jedoch die Erschöpfung für diesen Test deaktiviert.

**Annahmen:**

- Anzahl Personen: 1
- Geschwindigkeitsverteilung: die Person ist 25 Jahre alt. Es ist wichtig, die standardmäßige Altersverteilung von 14-80 zu überschreiben: Da die Strecke von einer einzigen Person gelaufen wird, entstehen große Unregelmäßigkeiten, wenn die Person in jedem Durchlauf ein unterschiedliches Alter besitzt, da dies unterschiedliche Startgeschwindigkeiten bedeutet.
- Anzahl der Durchläufe: 10
- Länge der Strecke: 40 Meter

**Ergebnisse:**

- durschnittliche Geschwindigkeit /m/s: ca. 1,25

Bis auf kleine Sprünge im Diagramm, die aus entwurfsrelevanten Gründen (wie etwa dem Trödeln) entstehen und keinen Einfluss auf den Durschnitt haben, bleibt die Geschwindigkeit konstant.

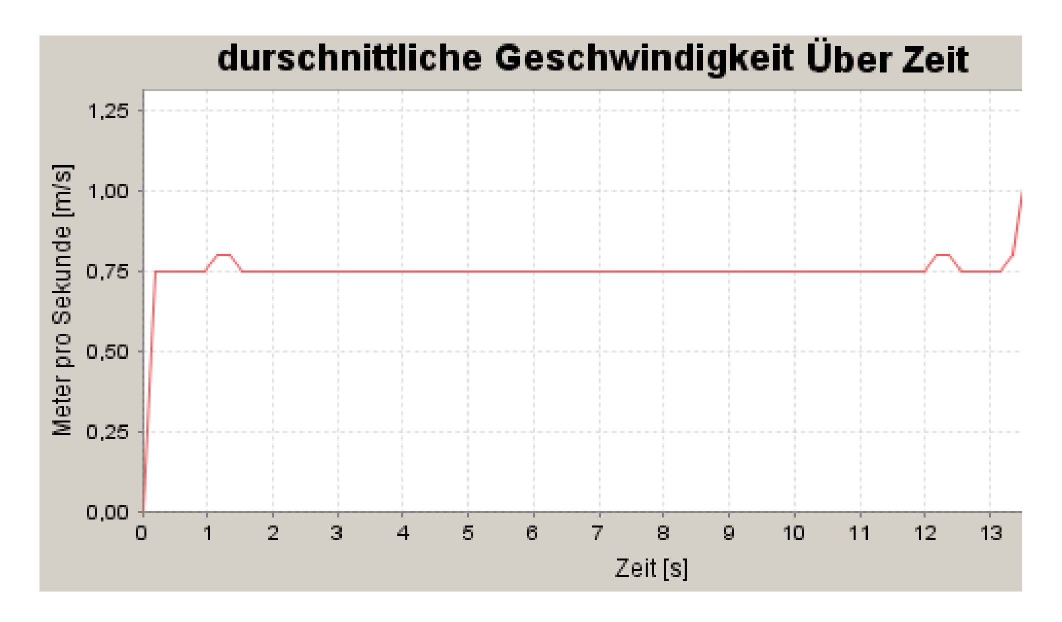



**Abbildung 8.2.:** Durschnittliche Geschwindigkeit über Zeit Rimea-Test 2

### 8.2.2. Test 2: Beibehalten der Geschwindigkeit treppauf

Ähnlich wie bei dem ersten Test geht es hier darum, das Beibehalten der Geschwindigkeit zu verifizieren, dieses Mal jedoch treppauf. Genau so wie beim ersten Test wurde der Erschöpfungseffekt deaktiviert.

- Anzahl Personen: 1
- Geschwindigkeitsverteilung: die Person ist 25 Jahre alt (vgl. Test 1)
- Anzahl der Durchläufe: 10
- Länge der Strecke: 10 Meter (treppauf entlang der Schräge der Treppe)

**Ergebnisse:**

- durschnittliche Geschwindigkeit /m/s: ca. 0,75

Die Geschwindigkeit ist wieder annähernd konstant. Man beachte jedoch, dass auf Treppen eine Geschwindigkeitsreduktion auftritt.

### 8.2.3. Test 3: Beibehalten der Geschwindigkeit treppab

Bei Test 3 handelt es sich um eine zu Test 2 ähnlichen Situation, nur mit umgekehrter Richtung. Es soll die Geschwindigkeit eines Individuum treppab

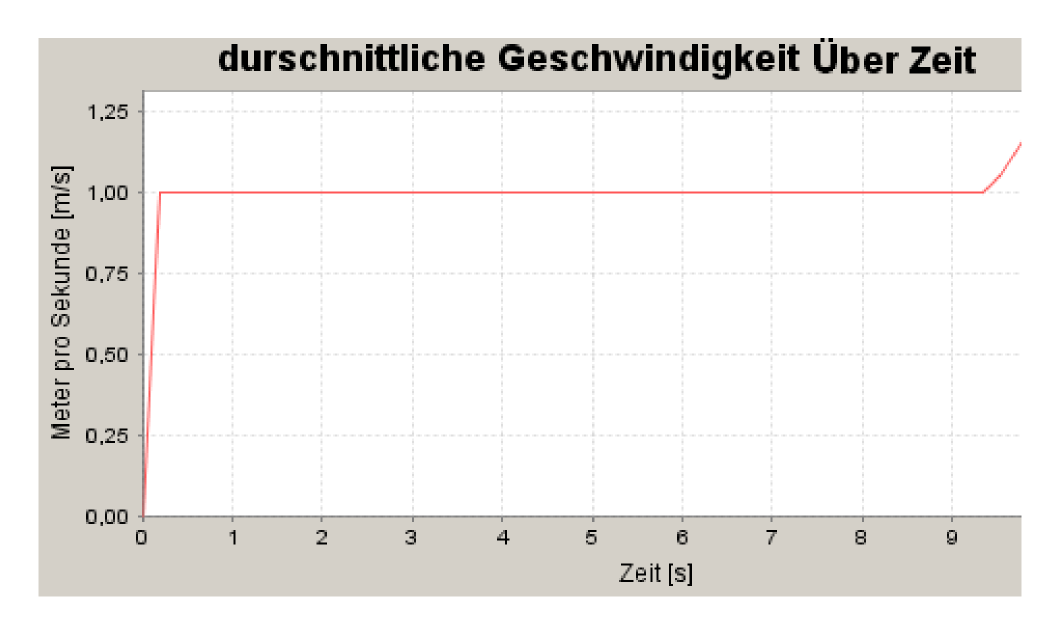



**Abbildung 8.3.:** Durschnittliche Geschwindigkeit über Zeit Rimea-Test 3

analysiert werden. ZET verhält sich analog zum zweiten Test mit wenigen Unterschieden:
Die Geschwindigkeit ist insgesamt höher als im zweiten Test, da treppab-gehen weniger anstrengend ist.

- Anzahl Personen: 1
- Geschwindigkeitsverteilung: die Person ist 25 Jahre alt (vgl. Test 1)
- Anzahl der Durchläufe: 10
- Länge der Strecke: 10 Meter (treppab entlang der Schräge der Treppe)

**Ergebnisse:**

- durschnittliche Geschwindigkeit /m/s: ca. 1,00

### 8.2.4. Test 4: Spezifischer Fluss durch einen Querschnitt

Bei diesem Test soll der spezifische Fluss an Personen durch einen Querschnitt gemessen werden. Unter dem spezifischen Fluss versteht man laut der Rimea-Richtline die "Anzahl Personen, die einen bestimmten Querschnitt pro Meter lichter Breite und pro Sekunde passieren". Dazu soll ein Gang mit "periodischen Randbedingungen" modelliert werden, was bedeutet, dass Personen, die den

Gang am Ende verlassen, diesen nach dem Verlassen wieder unmittelbar ohne Verzögerung am anderen Ende betreten sollen.

Die Modellierung der geforderten "periodischen Randbedingungen" ist mit dem ZET-Evakuierungstool leider konzeptbedingt nicht möglich. Um die Individuen dazu zu bringen, sich überhaupt zufallsbedingt in eine bestimmte Richtung zu bewegen, ist ein Potentialfeld notwendig, welches nicht ohne einen zugehörigen Ausgang existieren kann. Sobald die durch das Potentialfeld geleiteten Individuen den entsprechenden Ausgang erreichen, werden sie als *gerettet* eingestuft und können somit aus der Simulation entfernt werden, weil bereits gerettete Individuen für den weiteren Verlauf der aktuellen Simulation nicht mehr von Interesse sind. Sie können somit also den Gang nicht, wie in der Rimea-Richtlinie gefordert, am anderen Ende wieder betreten, sobald sie einen Ausgang erreicht haben. Auf der anderen Seite ist es aber aus oben genannten Gründen auch nicht möglich, die Individuen überhaupt in Bewegung zu setzen, ohne dass ein Ausgang existiert, zu dem sie sich hinbewegen. Dieser Test ist also nicht wie gefordert durchführbar.

### 8.2.5. Test 5: Reaktionsdauer

Bei diesem Test geht es darum, die Reaktionsdauer von Individuen zu testen. Unter der Reaktionsdauer wird hier die Zeit verstanden, die ein Individuum benötigt, um zu bemerken, dass das Gebäude, in dem es sich befindet, evakuiert werden soll. Technisch gesehen ist die Reaktionsdauer eines Individuums also die Zeit, die nach Beginn der Evakuierungssimulation verstreicht, bis sich das Individuum in Bewegung setzt und versucht, aus dem Gebäude zu flüchten.

Dabei soll hier ein Raum mit zehn Individuen modelliert werden, deren Reaktionsdauern gleichverteilt zwischen 10 und 100 Sekunden liegen, und bei der Evakuierungssimulation soll verifiziert werden, dass auch wirklich alle Individuen zu einem passenden Zeitpunkt starten.

Das ZET-Evakuierungstool ermöglicht es hier über verschiedene Regeln, entweder sofort alle Personen innerhalb eines Raumes zu informieren, sobald eine einzige Person alarmiert ist, oder jedem Individuum eine eigene unabhängige Reaktionszeit zuzuordnen, wie es für diesen Test verlangt wird.

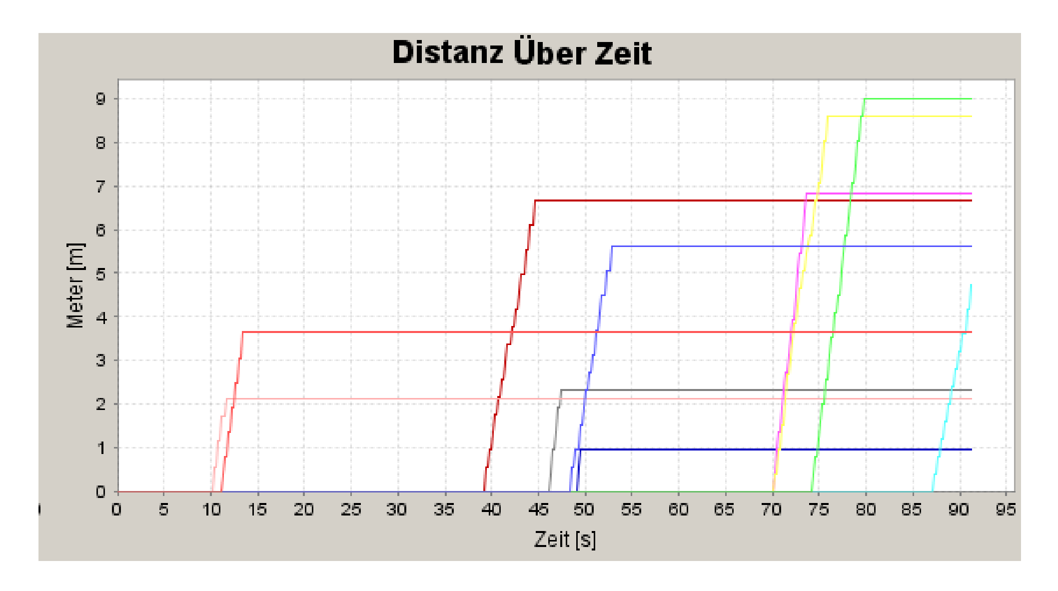



**Abbildung 8.4.:** Verteilung der Reaktionszeiten bei Test 5

Wie im Disatanz-Über-Zeit-Diagramm 8.4 für Test 5 zu sehen ist, starten die Personen jeweils erst nach einer individuellen Reaktionszeit. Diese sind wie zu sehen zwischen 10 und 100 Sekunden gleichverteilt.

### 8.2.6. Test 6: Bewegung um eine Ecke

Bei diesem Test geht es darum, dass zwanzig Personen erfolgreich eine nach links abbiegende Ecke passieren sollen, ohne dass es dabei zu Unregelmäßigkeiten wie dem Durchqueren von Wänden kommt. Der Testaufbau ist dabei im Screenshot, der in Abbildung 8.5 zu sehen ist, erkennbar. Die Personen befinden sich in der Ausgangssituation in dem blau markierten Belegunsbereich, der sich in der unteren linken Ecke des Raumes befindet. Ziel der Personen ist es, den grün markierten Evakuierungsbereich in der oberen rechten Ecke des Raumes zu erreichen. Dazu ist es notwendig, auf dem Weg die nach links abbiegende Ecke zu passieren.

In Abbildung 8.6 ist ein Screenshot der laufenden Simulation nach 5 Sekunden zu sehen und in Abbildung 8.7 ist ein Screenshot der laufenden Simulation nach 9 Sekunden zu sehen. In beiden Abbildungen ist deutlich erkennbar, dass alle Individuen die nach links abbiegende Ecke des Raumes korrekt passieren, ohne dass eine Person Wände durchquert. Dabei liegt die minimale Evakuierungszeit für eine Person bei etwa 9,3 Sekunden (vergleiche Abbildung 8.7),

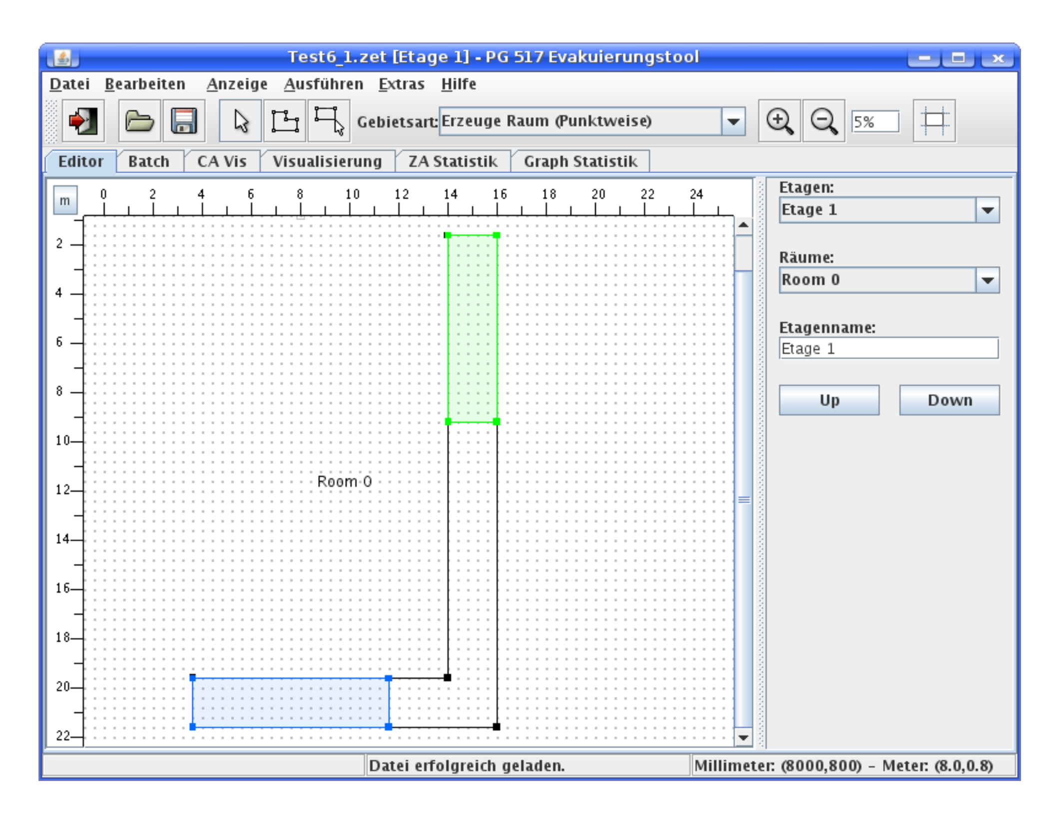



**Abbildung 8.5.:** Ausgangssituation für Rimea-Test 6

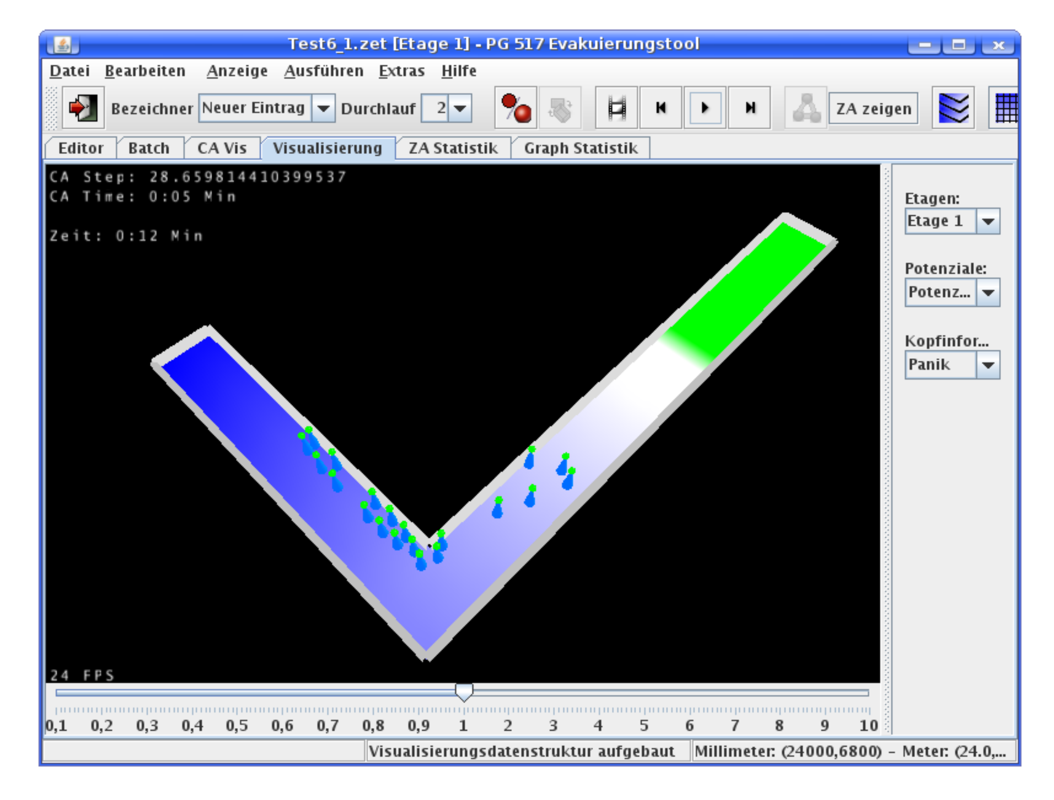



**Abbildung 8.6.:** Simulation des Rimea-Tests 6 nach 5 Sekunden

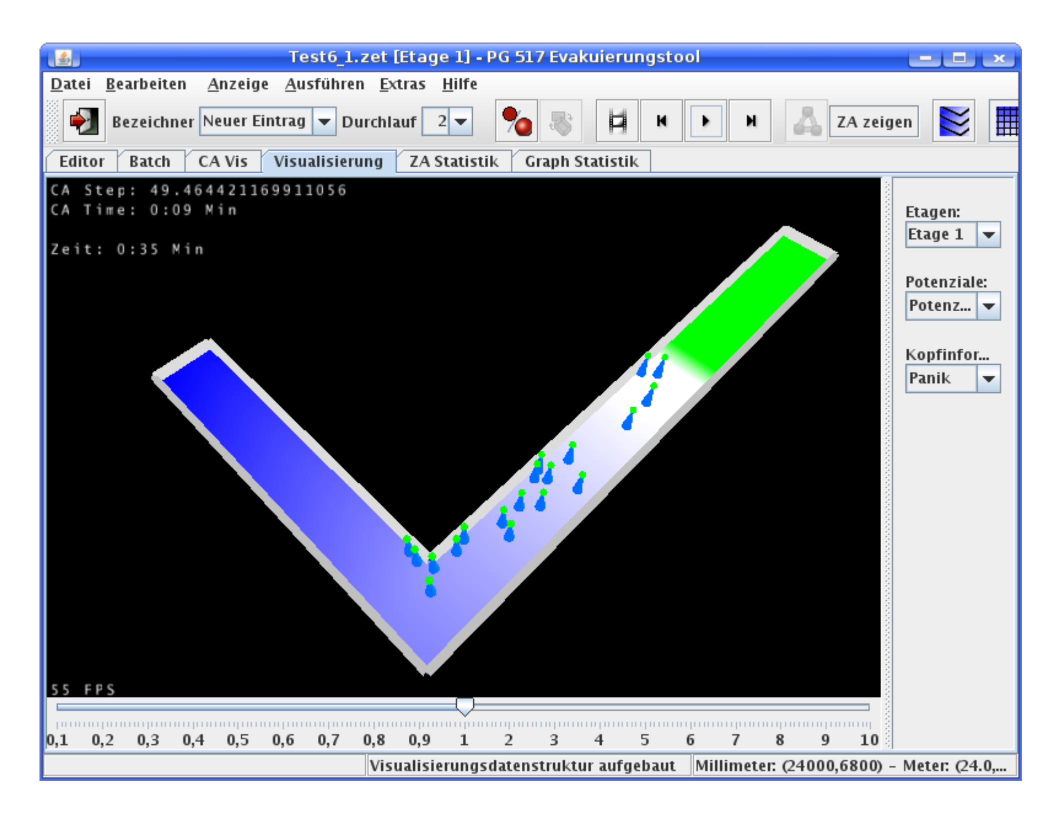



**Abbildung 8.7.:** Simulation des Rimea-Tests 6 nach 9 Sekunden

die maximale Evakuierungszeit einer Person beträgt etwa 22,7 Sekunden und die durchschnittliche Evakuierungszeit beträgt 14,8 Sekunden.

### 8.2.7. Test 7: Zuordnung der demographischen Parameter

Dieser Test dient dazu, zu überprüfen, ob die Gehgeschwindigkeiten korrekt verteilt werden und ob diese Geschwindigkeiten im Ablauf des zellulären Automaten eingehalten werden.

**Annahmen:**

- Anzahl Personen: 50
- Geschwindigkeitsverteilung: Personen im Alter von 14 bis 80, Erwartungswert 40, Varianz 30, Minimum:0.6, Maximum: 1.61

Die Ergebnisgeschwindigkeiten in Abbildung 8.8 sind die Wunschgeschwindigkeiten der Individuen. Diese werden durch weitere Einflüsse im Laufe der Evakuierung gesenkt oder erhöht. Solche Einflüsse sind z.B. Treppen, Verzögerungbereiche, Warten vor Hindernissen, Erschöpfung, Trödelfaktor und Panik.

**Ergebnisse**

| Person | Geschwindigkeit in m/s | Person | Geschwindigkeit in m/s | Person | Geschwindigkeit in m/s | Person | Geschwindigkeit in m/s | Person | Geschwindigkeit in m/s |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1,5 | 11 | 1,2 | 21 | 1,4 | 31 | 1,3 | 41 | 1,4 |
| 2 | 1,4 | 12 | 1,3 | 22 | 1,6 | 32 | 1,5 | 42 | 1,7 |
| 3 | 1,2 | 13 | 1,3 | 23 | 1,5 | 33 | 1,4 | 43 | 0,9 |
| 4 | 1,5 | 14 | 1,4 | 24 | 1,5 | 34 | 1,5 | 44 | 0,8 |
| 5 | 1,1 | 15 | 1,4 | 25 | 1,3 | 35 | 1,4 | 45 | 1,5 |
| 6 | 1,1 | 16 | 1,2 | 26 | 1,6 | 36 | 1,7 | 46 | 1,3 |
| 7 | 1,8 | 17 | 1,5 | 27 | 1,6 | 37 | 1,7 | 47 | 1,3 |
| 8 | 1,7 | 18 | 1,4 | 28 | 1,5 | 38 | 1,4 | 48 | 1,2 |
| 9 | 1,6 | 19 | 1,5 | 29 | 1,6 | 39 | 1,5 | 49 | 1,6 |
| 10 | 1,0 | 20 | 1,5 | 30 | 0,9 | 40 | 1,7 | 50 | 1,1 |

**Abbildung 8.8.:** Ergebnisgeschwindigkeiten in Test 7

Die durchschnittliche Geschwindigkeit der Individuen bei einer Evakuierung ohne Hindernisse, Verzögerungsbereiche und Treppen sieht z.B. wie in Abbildung 8.9 abgebildet aus. Die maximale Geschwindigkeit von 1,61 m/s bei unter 30 jährigen wird nicht überschritten. Die minimale Geschwindigkeit liegt bei 0,6 m/s und die durchschnittliche bei 1,27 m/s. Damit entsprechen die Geschwindigkeiten den im Rimea-Protokoll vorgegebenen.

## 8.3. Funktionale Verifizierung

„Funktionale Verifizierung schließt ein zu überprüfen, dass das Modell die Fähigkeit besitzt, den Bereich der für die Simulation notwendigen Möglichkeiten abzudecken. Diese Anforderung ist aufgabenspezifisch. Um funktionale Verifizierung zu erfüllen, müssen die Entwickler des Modells in verständlicher Weise den vollen Bereich der Möglichkeiten des Modells und der inhärenten Annahmen darstellen und eine Anleitung für den korrekten Gebrauch dieser Möglichkeiten zur Verfügung stellen. Informationen sollen in der technischen Do-

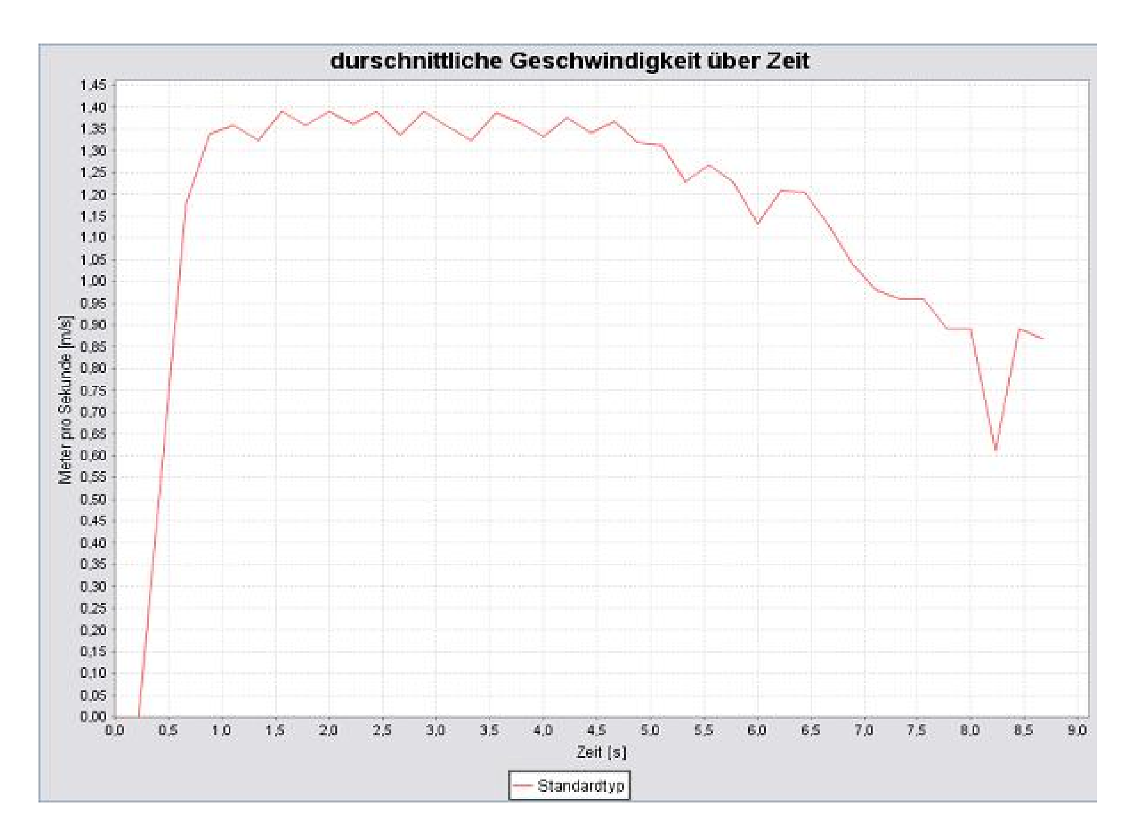



**Abbildung 8.9.:** Durchschnittliche Geschwindigkeit in Test 7.

kumentation der Software leicht zugänglich sein.“

### 8.3.1. Test 8: Parameteranalyse

Dieser Test soll es ermöglichen, die Auswirkungen der verschiedenen Parameter auf die Gesamtentfluchtungsdauer (in Sekunden) aufzuzeigen, wenn jeweils ein Parameter variiert und die Restlichen auf feste Standardwerte gesetzt werden. Die Variation geschieht zum einen im Rahmen fester Werte und zum anderen gemäß verschiedener Normalverteilungen um unterschiedliche Mittelwerte. Die untersuchten Parameter sind: **Alter**, **Entschlossenheit**, **Panik** und **Ortskenntnis**. Es wurden für jeden Parameter verschiedene feste Werte und Normalverteilungen betrachtet und für diese `ParameterSets` jeweils 10 Simulationen durchgeführt, über die gemittelt wurde. Soweit nicht anders angegeben, haben die Parameter folgende Standardwerte:

- Alter = 20,
- Entschlossenheit = 0,3,

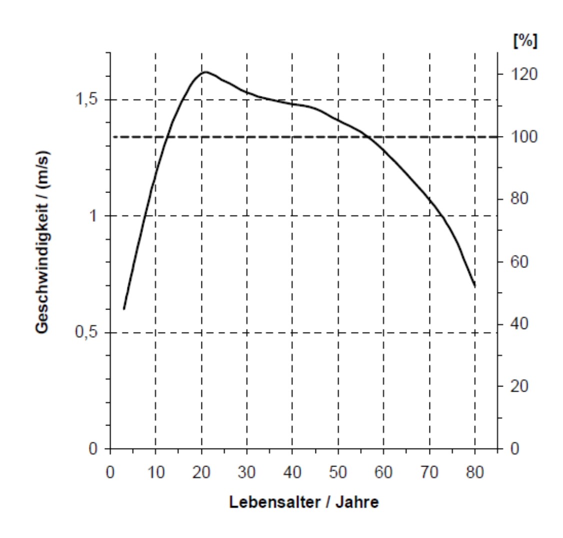



**Abbildung 8.10.:** Geschwindigkeitsverteilung in Abhängigkeit des Alters

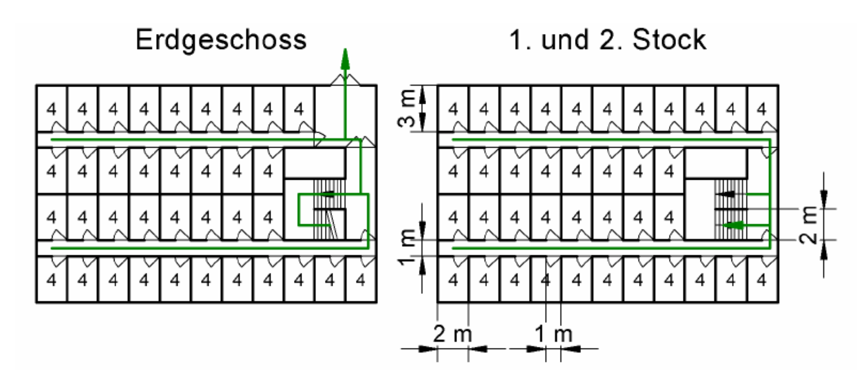



**Abbildung 8.11.:** Grundriss aus der RiMEA-Richtlinie

- Panik = 0,5
- Ortskenntnis = 0,3.

Der Parameter *Alter* wirkt sich übrigens gemäß der RiMEA-Richtlinie auf die Geschwindigkeiten der Individuen aus (vgl. Abb. 8.10). Die Simulationen wurden auf dem in Abbildung 8.11 abgebildeten dreistöckigen Grundriss ausgeführt. Die quantitativen Simulationsergebnisse sind in den Abbildungen 8.12 - 8.27 dargestellt. Qualitativ lässt sich folgendes aussagen:

- Der Graph der Evakuierungszeit verhält sich qualitativ bei festem Alter praktisch wie der an der X-Achse gespiegelte Graph der altersabhängigen

| Alter | EvacTime-max | EvacTime-min | EvacTime-mean |
|---|---|---|---|
| 5 | 193 | 184 | 187 |
| 20 | 137 | 124 | 128 |
| 55 | 154 | 142 | 146 |
| 80 | 192 | 180 | 185 |

**Abbildung 8.12.:** Alter – feste Werte

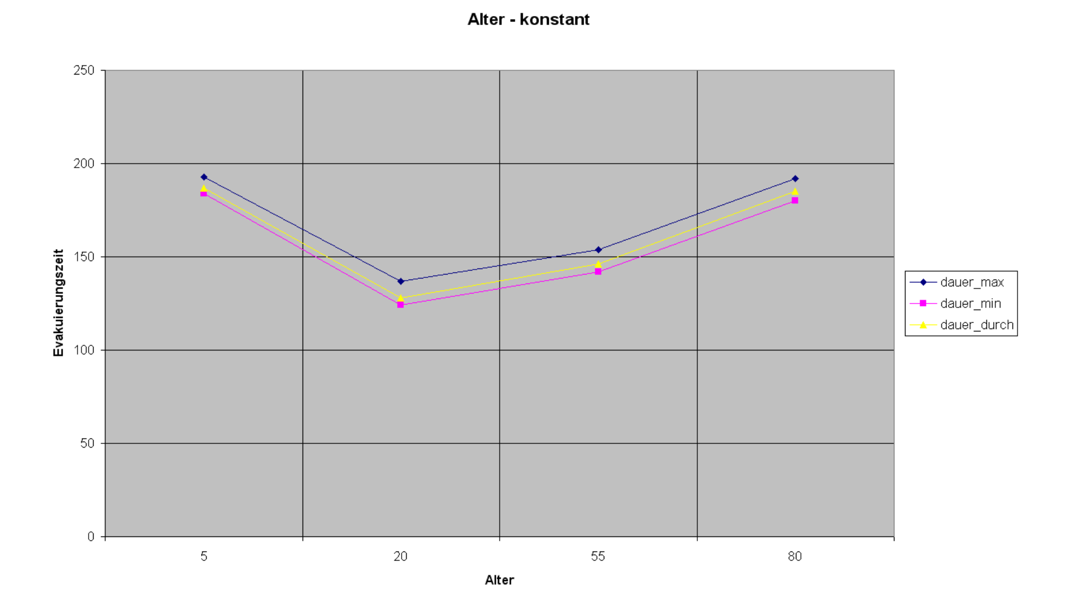



**Abbildung 8.13.:** Alter – konstant

Geschwindigkeiten (vgl. Abb. 8.10). Und auch für normalverteiltes Alter sind die Individuen mittleren Alters am schnellsten.

- Die Graphen der Evakuierungszeit bzgl. Entschlossenheit zeigen auf, dass entschlossene Personen schneller ihr Ziel erreichen, da sie mit geringerer Wahrscheinlichkeit ihre Route ändern und sich somit schnellstmöglich zum Ausgang bewegen.

- Panik hat nur geringen Einfluss auf die Evakuierungszeit, wenngleich zu beobachten ist, dass geringe Panik zu einer leicht schnelleren Evakuierung führt.

- Die Vertrautheit hat in diesem Beispiel praktisch keinen Einfluss auf die Evakuierungszeit, da das betrachtete Gebäude nur einen Ausgang hat und somit eine höhere Ortskenntnis keinerlei Vorteile einbringt.

| Alter | EvacTime-max | EvacTime-min | EvacTime-mean |
|---|---|---|---|
| 5-20 | 158 | 143 | 150 |
| 20-55 | 142 | 128 | 134 |
| 55-80 | 142 | 129 | 135 |
| 5-80 | 166 | 150 | 159 |

**Abbildung 8.14.:** Alter – normalverteilt.

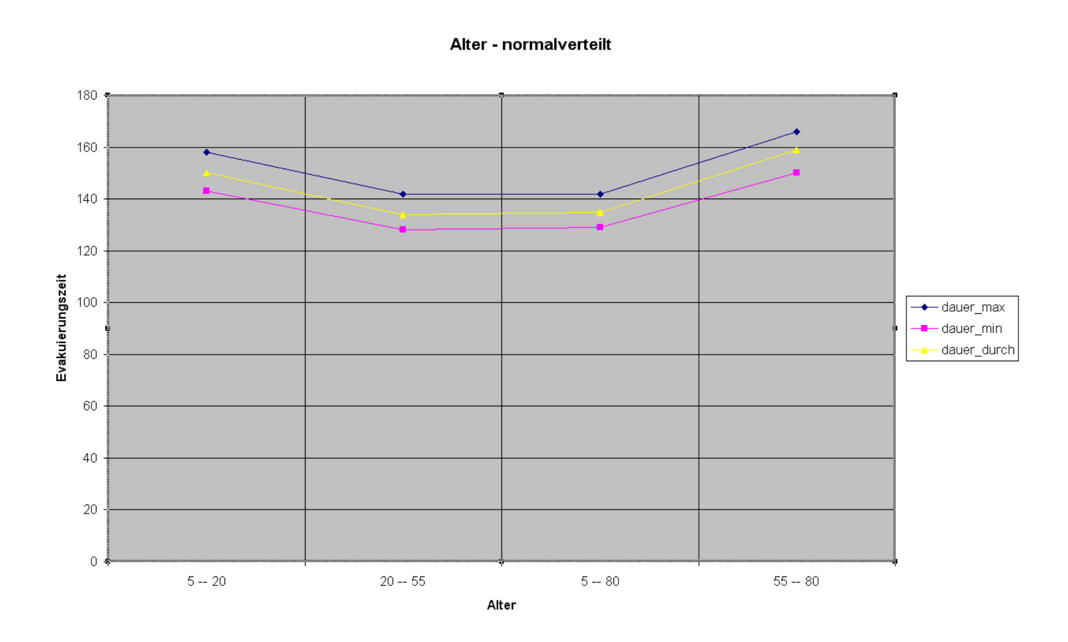



**Abbildung 8.15.:** Alter – normalverteilt.

| Entschlossenheit | EvacTime-max | EvacTime-min | EvacTime-mean |
|---|---|---|---|
| 0,01 | 140 | 125 | 130 |
| 0,25 | 135 | 124 | 126 |
| 0,5 | 130 | 122 | 123 |
| 0,75 | 135 | 124 | 125 |
| 1 | 133 | 122 | 124 |

**Abbildung 8.16.:** Entschlossenheit – feste Werte

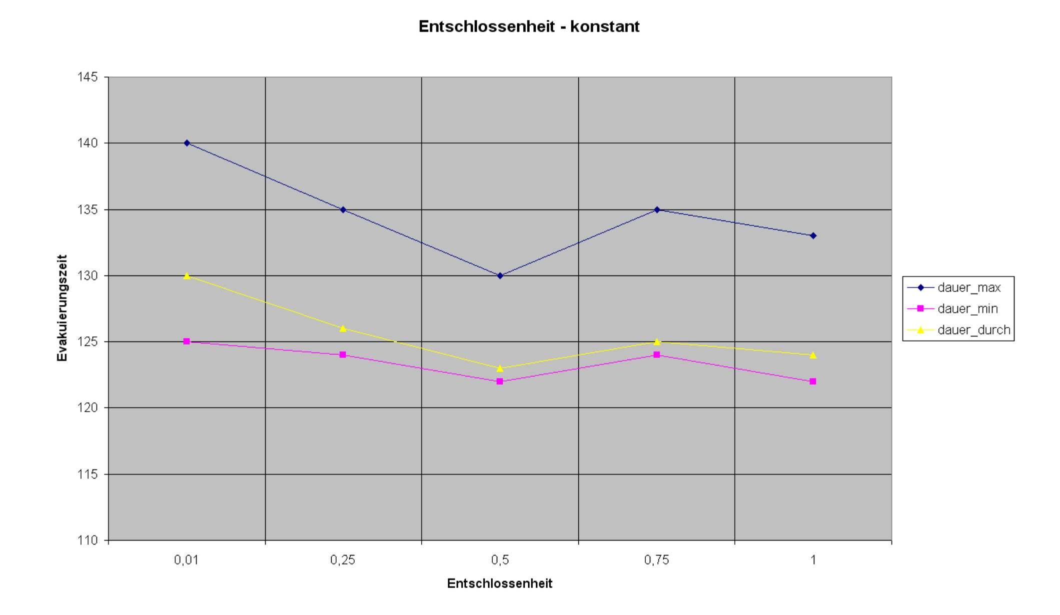



**Abbildung 8.17.:** Entschlossenheit - feste Wert

| Entschlossenheit | EvacTime-max | EvacTime-min | EvacTime-mean |
|---|---|---|---|
| 0,01-0,5 | 131 | 125 | 127 |
| 0,5-1 | 134 | 122 | 127 |
| 0,01-1 | 135 | 123 | 125 |

**Abbildung 8.18.:** Entschlossenheit – normalverteilt

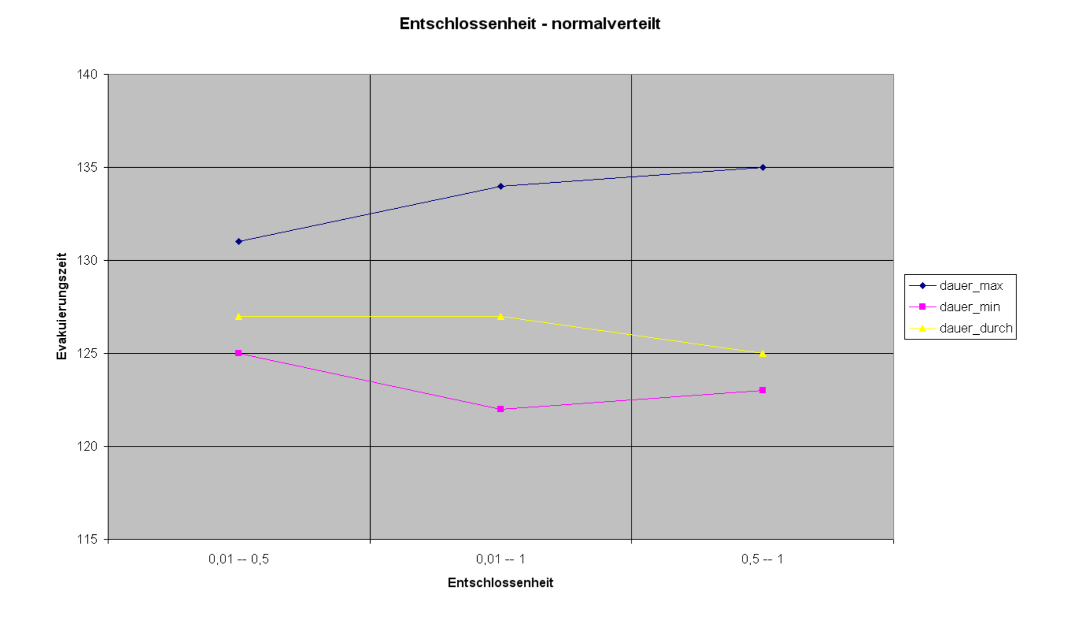



**Abbildung 8.19.:** Entschlossenheit – normalverteilt

| Panik | EvacTime-max | EvacTime-min | EvacTime-mean |
|---|---|---|---|
| 0,01 | 127 | 114 | 117 |
| 0,25 | 143 | 128 | 131 |
| 0,5 | 135 | 122 | 127 |
| 0,75 | 135 | 123 | 125 |
| 1 | 132 | 120 | 126 |

**Abbildung 8.20.:** Panik – feste Werte

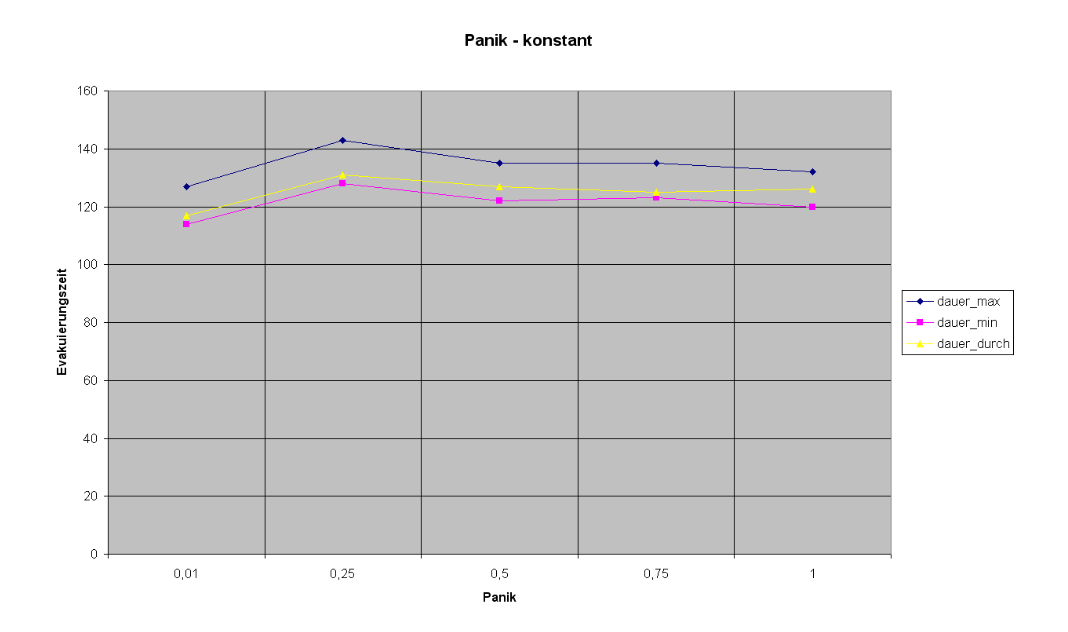



**Abbildung 8.21.:** Panik – feste Werte

| Panik | EvacTime-max | EvacTime-min | EvacTime-mean |
|---|---|---|---|
| 0,01-0,5 | 135 | 126 | 130 |
| 0,5-1 | 133 | 123 | 130 |
| 0,01-1 | 130 | 123 | 125 |

**Abbildung 8.22.:** Panik – normalverteilt

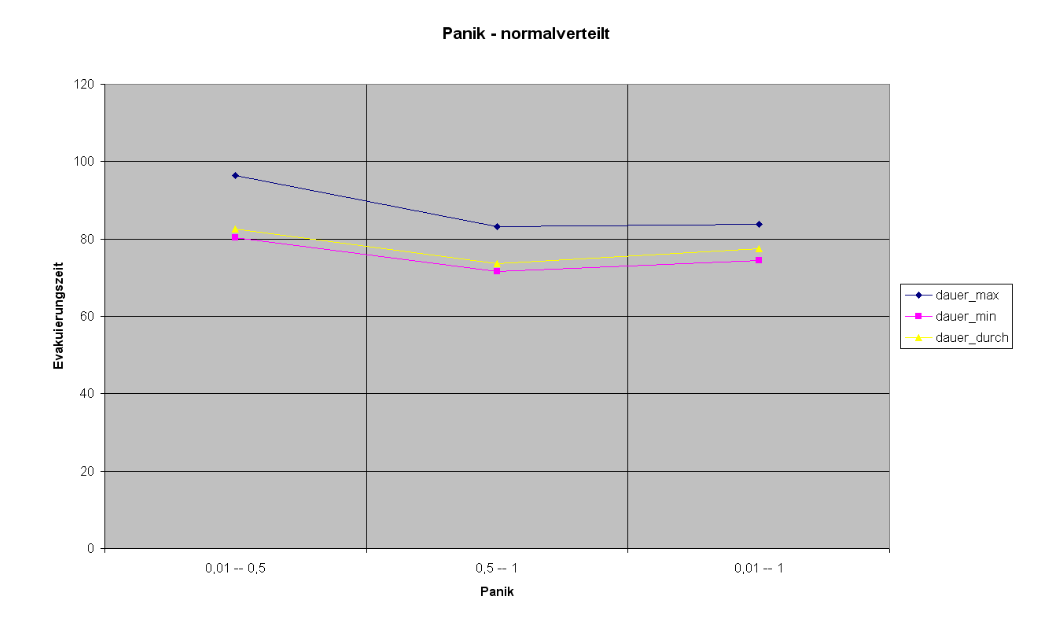



**Abbildung 8.23.:** Panik – normalverteilt

| Ortskenntnis | EvacTime-max | EvacTime-min | EvacTime-mean |
|---|---|---|---|
| 0,01 | 135 | 121 | 126 |
| 0,25 | 141 | 123 | 128 |
| 0,5 | 137 | 124 | 127 |
| 0,75 | 136 | 124 | 127 |
| 1 | 136 | 121 | 126 |

**Abbildung 8.24.:** Ortskenntnis – feste Werte

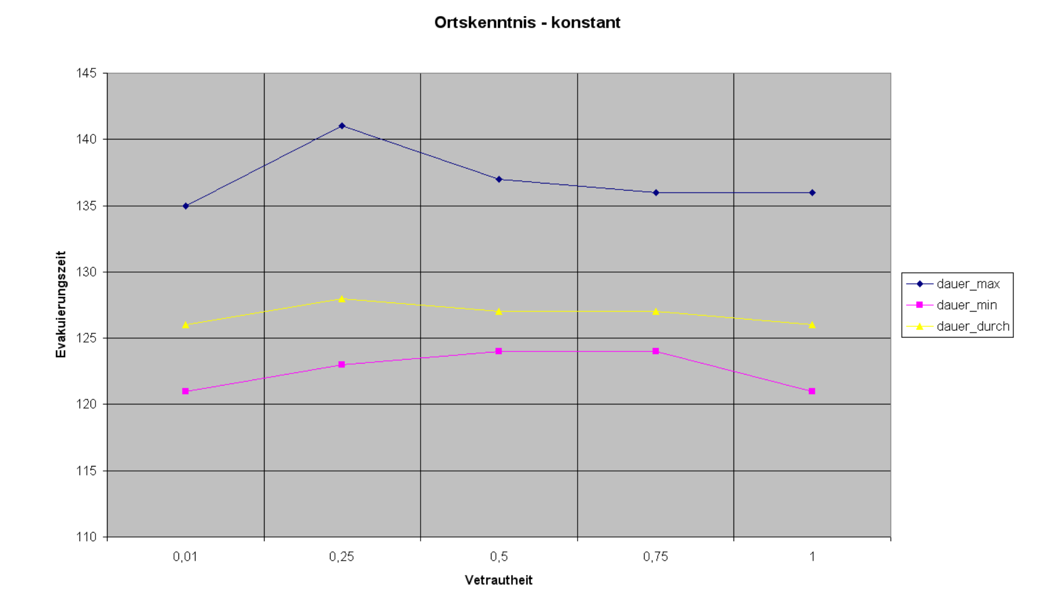



**Abbildung 8.25.:** Ortskenntnis – feste Werte

| Ortskenntnis | EvacTime-max | EvacTime-min | EvacTime-mean |
|---|---|---|---|
| 0,01-0,5 | 135 | 124 | 127 |
| 0,5-1 | 132 | 122 | 126 |
| 0,01-1 | 135 | 124 | 127 |

**Abbildung 8.26.:** Ortskenntnis – normalverteilt

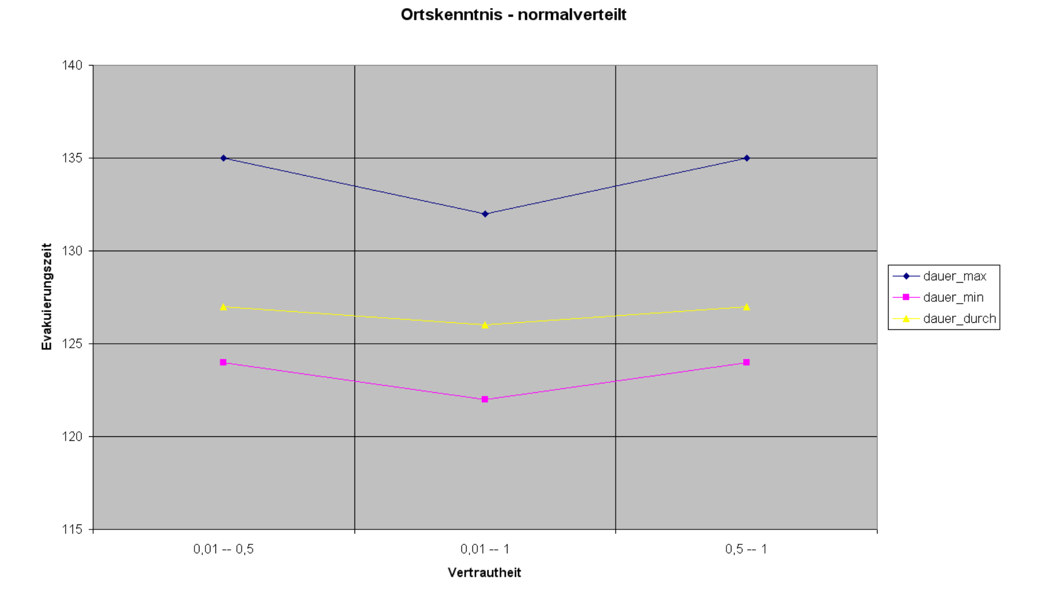



**Abbildung 8.27.:** Ortskenntnis – normalverteilt

## 8.4. Qualitative Verifizierung

„Die dritte Form der Modellvalidierung betrifft die Übereinstimmung des vorhergesagten menschlichen Verhaltens mit sachkundigen Erwartungen. Obwohl dies nur eine qualitative Form der Verifizierung darstellt, ist sie nichtsdestoweniger wichtig, da sie zeigt, dass die in dem Modell eingebauten Verhaltensweisen in der Lage sind, realistisches Verhalten zu erzeugen.“

### 8.4.1. Test 9: Menschenmenge verlässt großen Raum

In diesem Test wird ein 30mx20m großer Raum betrachtet. (siehe Abbildung 8.28 Im ersten Schritt verfügt der Raum über vier Türen und es sollen 1000 Personen evakuiert werden. Anschließend werden im zweiten Schritt 2 Türen versperrt und eine erneute Evakuierung der 1000 Personen durchgeführt. Die Evakuierungszeit im zweiten Schritt sollte ca. doppelt so groß sein.
**Annahmen:**

- Anzahl Personen: 1000
- Geschwindigkeitsverteilung: Personen im Alter von 14 bis 80, Erwartungswert 40, Varianz 25, Minimum:0.6, Maximum: 1.61
- Anzahl der Durchläufe: 10

**Ergebnisse:**

- Schritt 1, Dauer minimal /s: 95
- Schritt 2, Dauer minimal /s: 170
- Schritt 1, Dauer mittel /s: 98
- Schritt 2, Dauer mittel /s: 209
- Schritt 1, Dauer maximal /s: 107
- Schritt 2, Dauer maximal /s: 216

Die Ergebnisgeschwindigkeiten in Schritt 2 haben sich praktisch verdoppelt. Somit ist der vom Test gewünschte Effekt eingetreten.

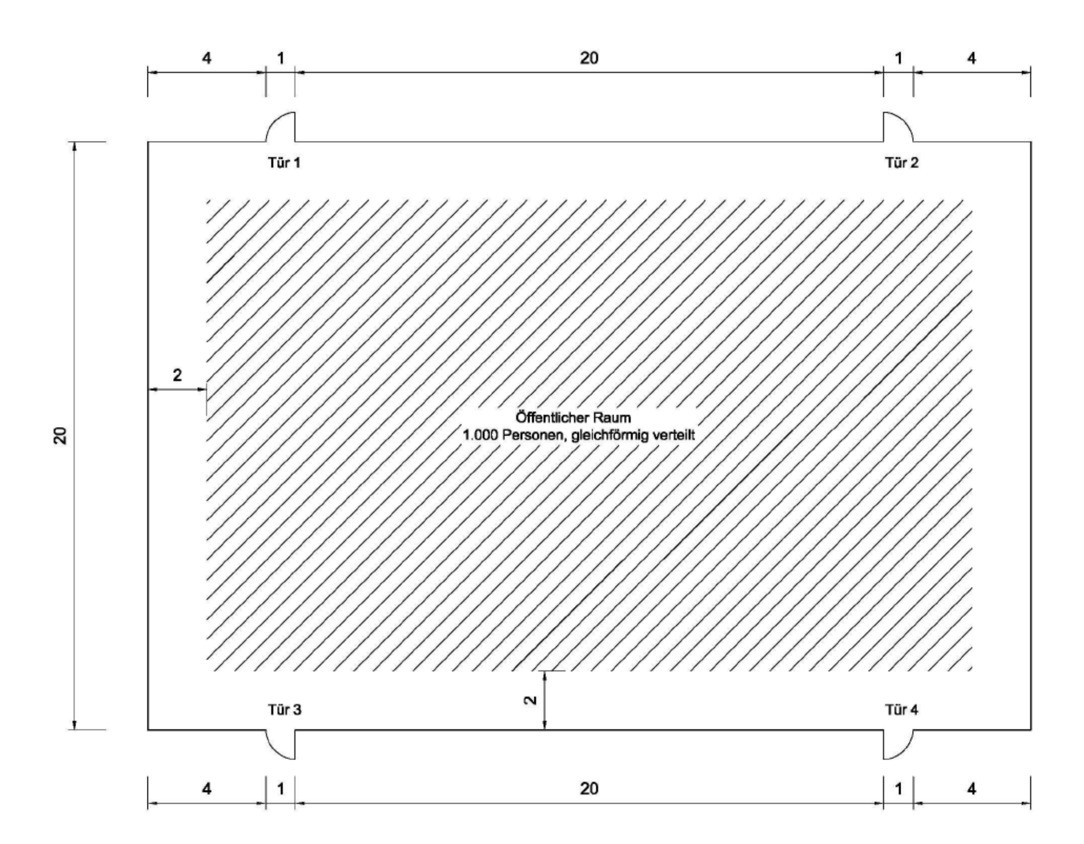



**Abbildung 8.28.:** Grundriss des Raumes für Test 9

### 8.4.2. Test 10: Zuweisung von Rettungswegen

In diesem Test wird untersucht, ob Personen den zugewiesenen nächsten Ausgang nehmen. Dazu betrachen wir ein Gebäude mit 12 Räumen und 2 Ausgängen wie in Abbildung 8.29 zu sehen. Das erwartete Ergebnis ist, dass die Personen in den Räumen 1, 2, 3, 4, 7, 8, 9 und 10 den Hauptausgang nehmen, während die Personen in den Räumen 5, 6, 11, 12 das Gebäude durch den sekundären Ausgang verlassen.

**Annahmen:**

- Anzahl Personen: 23
- Geschwindigkeitsverteilung: Personen im Alter von 14 bis 80, Erwartungswert 40, Varianz 25, Minimum: 0.6, Maximum: 1.61
- Anzahl der Durchläufe: 10

**Ergebnisse:**

- Dauer minimal /s: 12

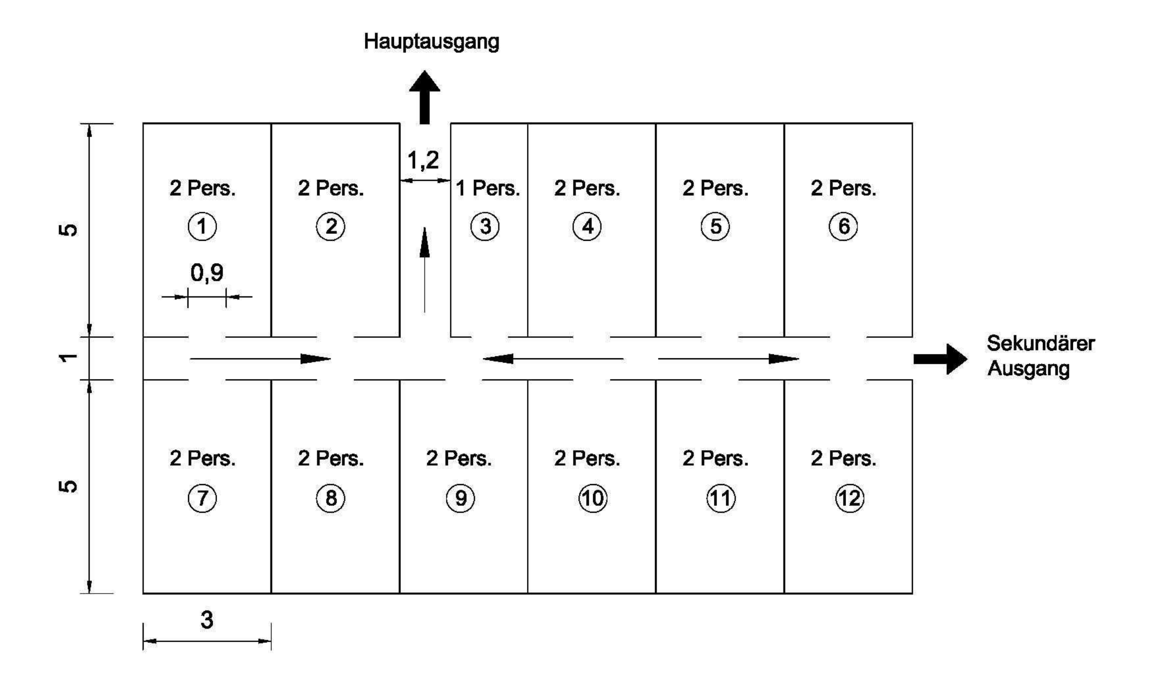



**Abbildung 8.29.:** Gang mit angrenzenden Räumen (Einheit: m).

- Dauer mittel /s: 13
- Dauer maximal /s: 21
- Personenanzahl am Hauptausgang: 15
- Personenanzahl am sekundären Ausgang: 8

Die geforderte Verteilung hat sich demnach eingestellt.

### 8.4.3. Test 11: Wahl des Rettungsweges

In diesem Test soll das Verhalten einer Population bestehend aus 1000 Personen in einem großen Raum mit zwei Ausgängen überprüft werden (vgl. Abbildung 8.30). Das erwartete Ergebnis ist, dass die Personen den näheren Ausgang bevorzugen und es deshalb dort zu einer Staubildung kommt. Einzelne Personen sollen dann den zweiten Ausgang wählen.

**Annahmen:**

- Anzahl Personen: 1000
- Geschwindigkeitsverteilung: Personen im Alter von 14 bis 18, Erwartungswert 16, Varianz 1, Minimum: 0.6, Maximum: 1.61

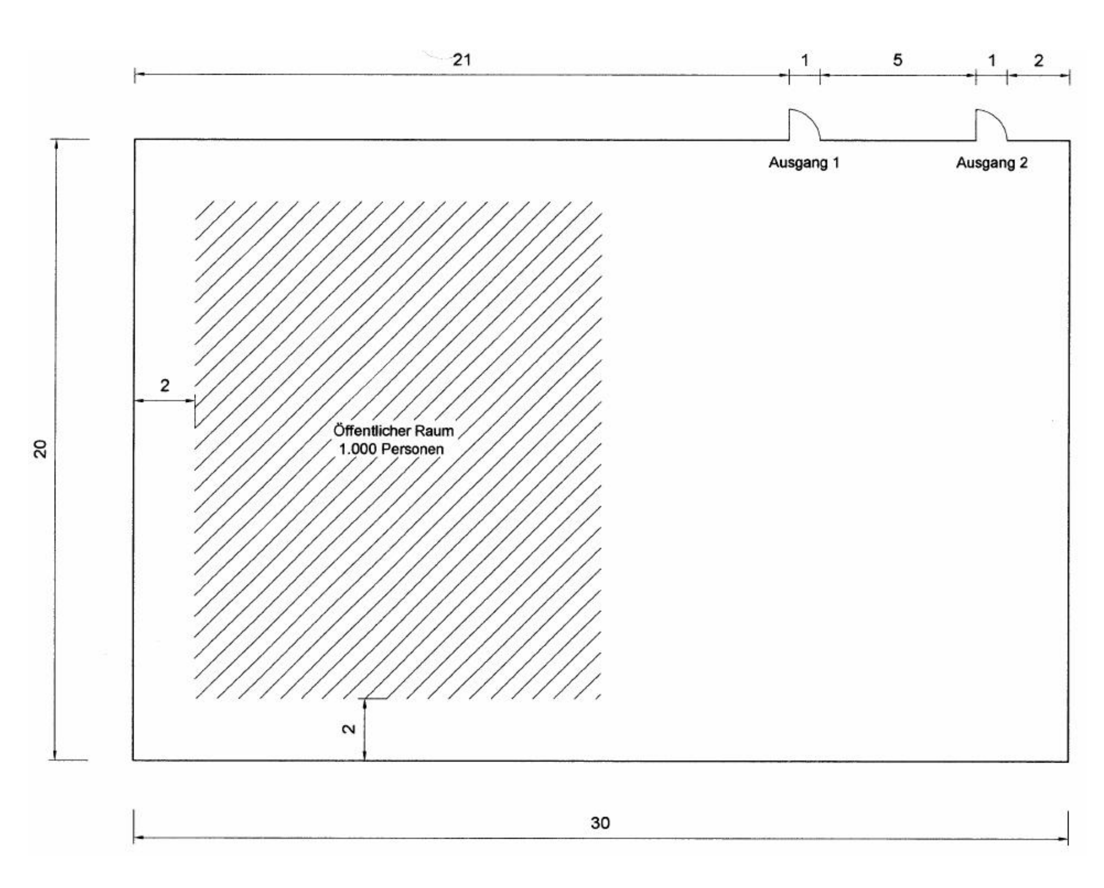



**Abbildung 8.30.:** Verlassen eines Raumes über zwei Ausgänge (Einheit: m).

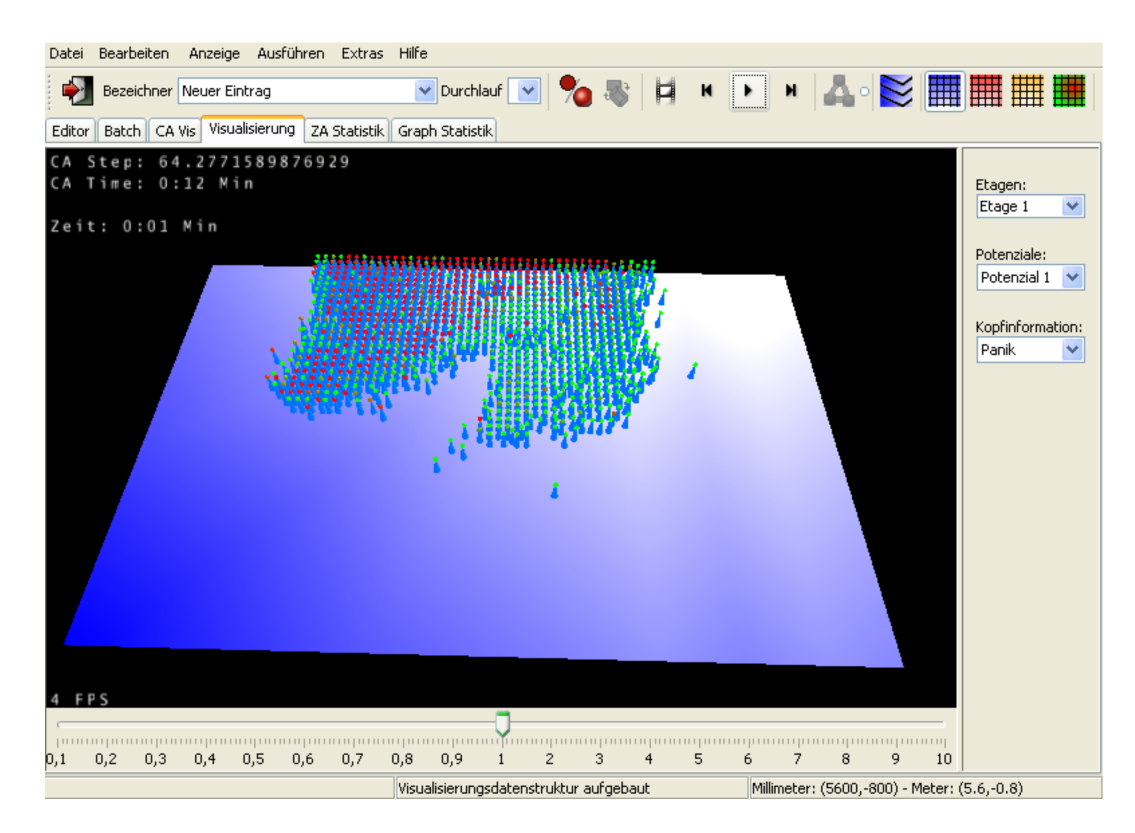



**Abbildung 8.31.:** Test 11 nach 12 Sekunden.

- Anzahl der Durchläufe: 10

**Ergebnisse:**

- Dauer minimal /s: 117
- Dauer mittel /s: 120
- Dauer maximal /s: 131
- Personenanzahl Ausgang 1 (Mittel): 542
- Personenanzahl Ausgang 2 (Mittel): 468

Wie zuvor erwartet, wird der nähere Ausgang bevorzugt. Dort entsteht auch die erwartete Stauung (vgl. Abbildung 8.31). Der weiter entfernte Ausgang wird auch von einigen Indidividuen gewählt, wie in Abbildung 8.32 zu sehen ist.

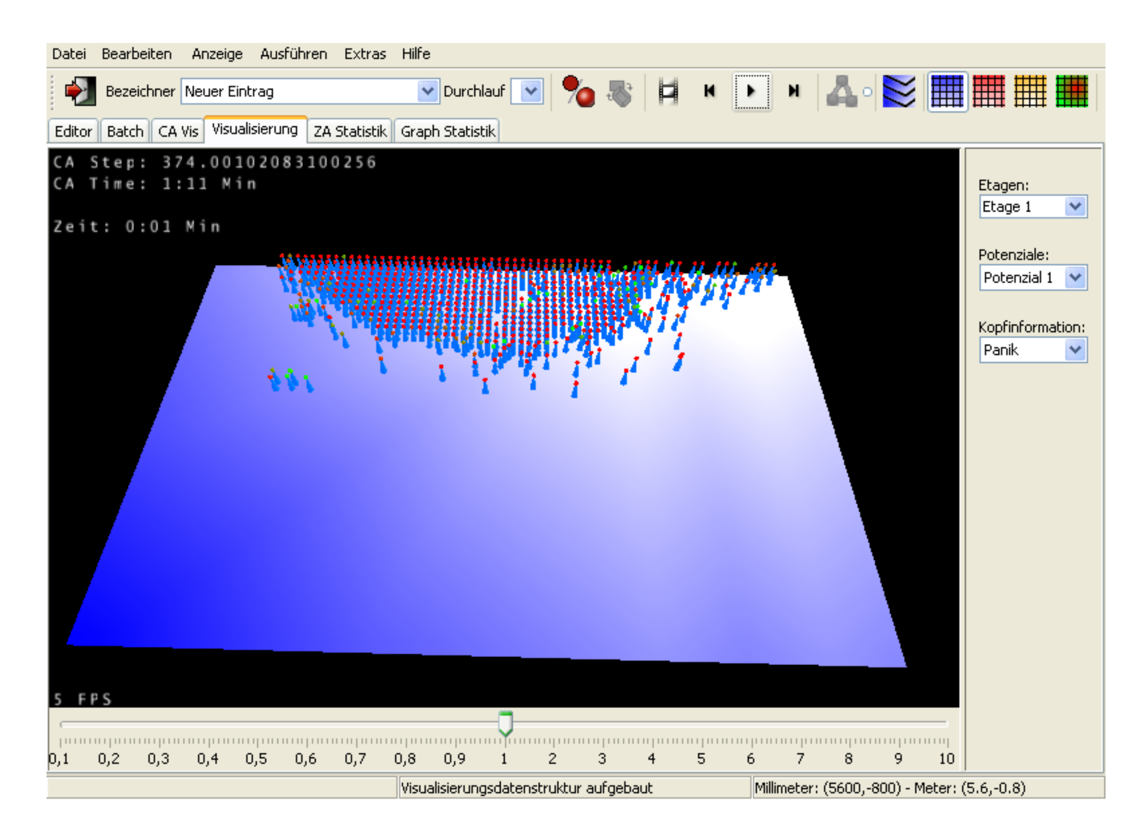



**Abbildung 8.32.:** Test 11 nach 71 Sekunden.

### 8.4.4. Test 12: Auswirkungen von Engstellen

In diesem Test soll überprüft werden, ob eine Stauung an einer Engstelle auftritt. Dafür betrachten wir zwei Räume, die mit einem Gang verbunden sind. In dem einen Raum befindet sich die Population von 150 Personen, in dem anderen Raum befindet sich der Ausgang, wie in Abbildung 8.33 zu sehen. Das erwartete Ergebnis ist, dass in Raum 1 vor der Engstelle eine Stauung entsteht.

**Annahmen:**

- Anzahl Personen: 150
- Geschwindigkeitsverteilung: Personen im Alter von 14 bis 80, Erwartungswert 40, Varianz 25, Minimum: 0.6, Maximum: 1.61
- Anzahl der Durchläufe: 10

**Ergebnisse:**

- Dauer minimal /s: 51

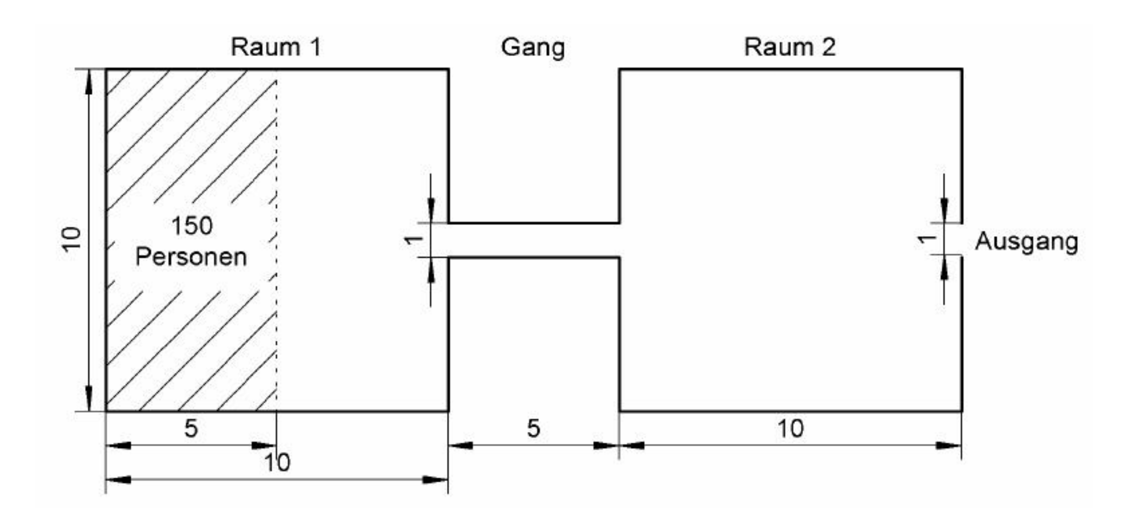



**Abbildung 8.33.:** Die Auswirkung der Engstelle führt zu einer Staubildung vor dem Gang wodurch ein Stau vor dem Ausgang vermieden wird.

- Dauer mittel /s: 52
- Dauer maximal /s: 59

Man kann an den Abbildungen 8.34 und 8.35 erkennen, dass das erwartete Ergebnis eingetreten ist. In Raum 1 bildete sich durch die Engstelle ein Stau (vgl. Abbildung 8.34). Durch den verringerten Personenfluss entstand kein Stau vor dem Ausgang in Raum 2, wie in Abbildung 8.35 zu sehen ist.

### 8.4.5. Test 13: Stau vor einer Treppe

In diesem Test soll der Einfluss von Treppen untersucht werden. Dafür betrachten wir einen Raum und einen angrenzenden Gang, der zu einem Ausgang führt. In diesem Gang befindet sich noch eine Treppe, wie in Abbildung 8.36 zu sehen ist. Das erwartete Ergebnis ist, dass zunächst eine Stauung am Ausgang des Raumes entsteht. Mit der Zeit soll dann eine Stauung am Fuß der Treppe entstehen.

**Annahmen:**

- Anzahl Personen: 150
- Geschwindigkeitsverteilung: Personen im Alter von 14 bis 80, Erwartungswert 40, Varianz 25, Minimum: 0.6, Maximum: 1.61

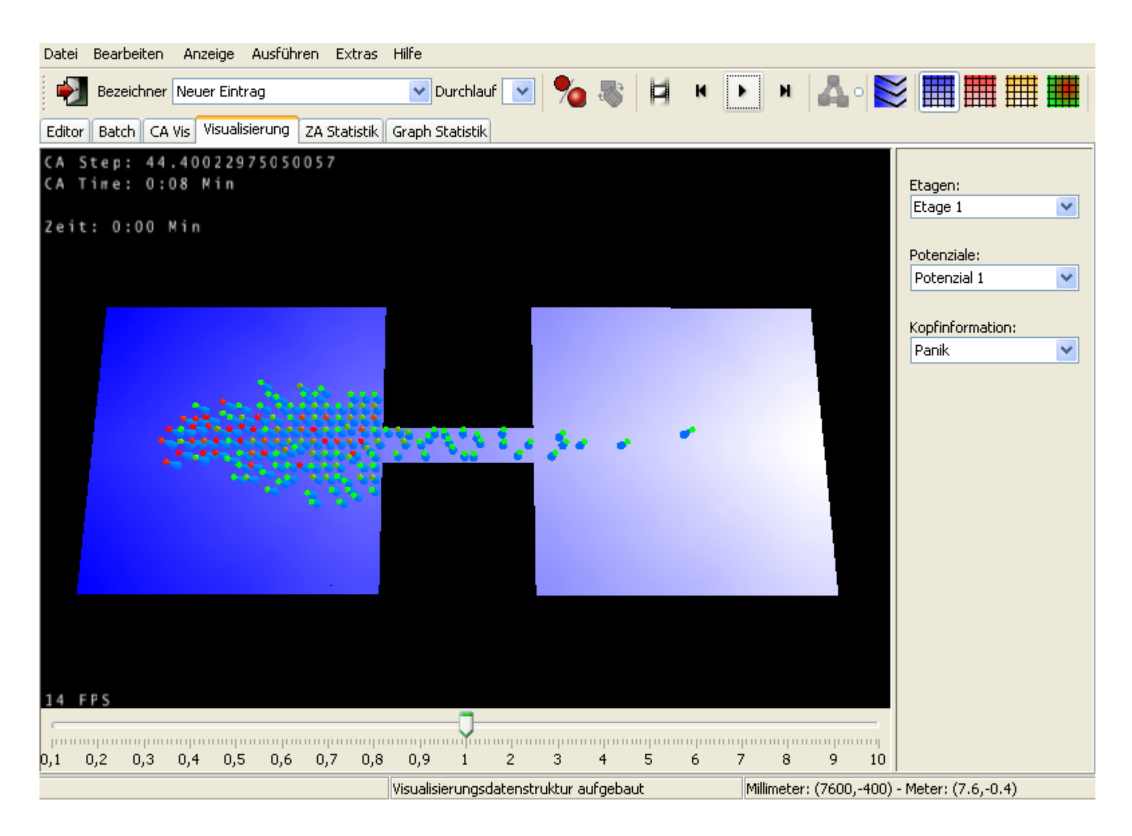



**Abbildung 8.34.:** Test 12 nach 8 Sekunden.

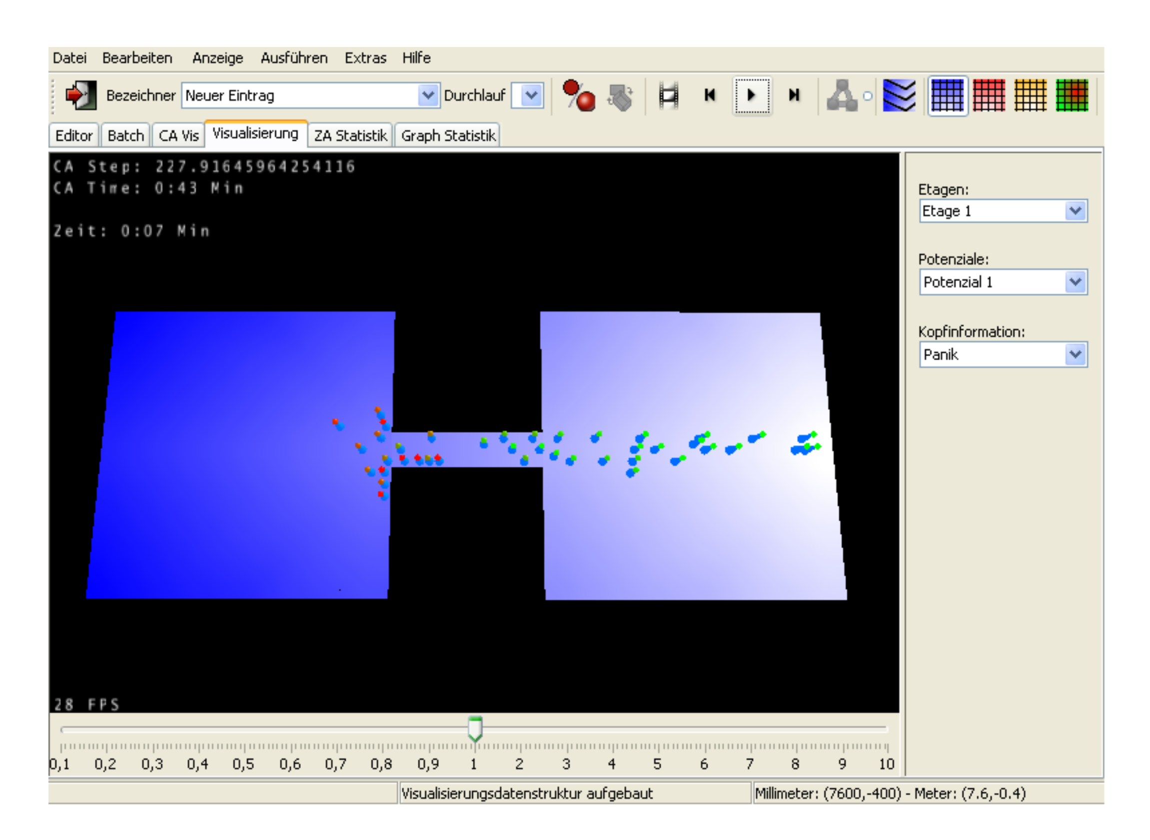



**Abbildung 8.35.:** Test 12 nach 43 Sekunden.

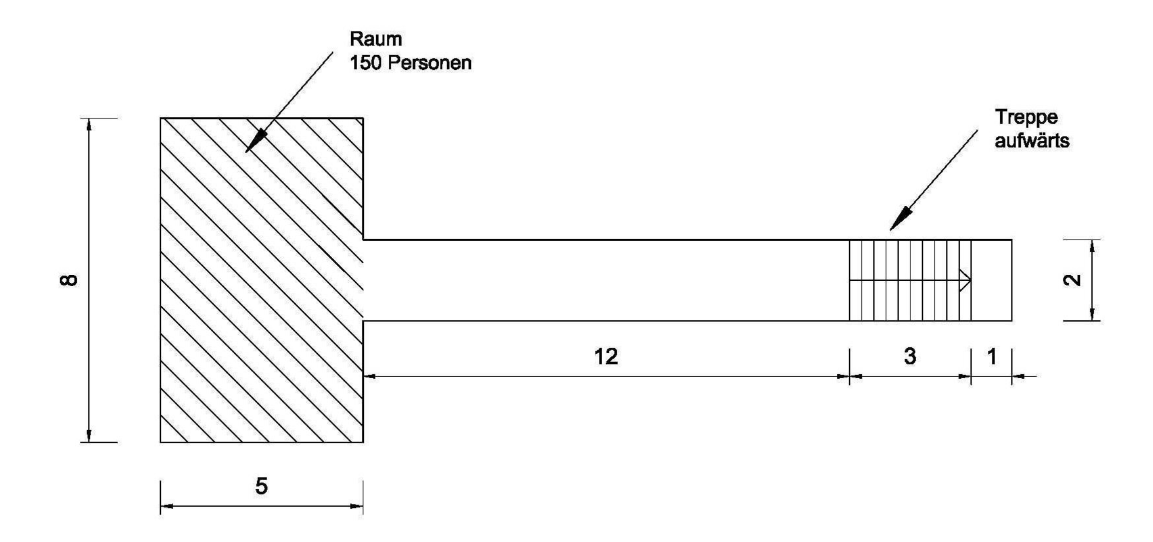



**Abbildung 8.36.:** Rettungsweg über Treppe (Einheit: m).

- Anzahl der Durchläufe: 10

**Ergebnisse:**

- Dauer minimal /s: 38
- Dauer mittel /s: 41
- Dauer maximal /s: 45

Auch hier sind die erwarteten Ergebnisse eingetreten. Zu Beginn enstand eine Stauung vor dem Ausgang des Raumes (vgl. Abbildung 8.37). Der Stau sorgte im Folgenden für einen verringerten Personenfluss im Gang. Dadurch entstand nur ein kleiner Stau vor der Treppe (vgl. Abbildung 8.38).

## 8.5. Quantitative Verifizierung

„Quantitative Verifizierung beinhaltet den Vergleich von Modellvorhersagen aus Evakuierungsübungen. Zum jetzigen Entwicklungszeitpunkt sind nicht genügend zuverlässige experimentelle Daten vorhanden, um eine gründliche quantitative Entfluchtungsmodellen zu erlauben. Solang bis solche Daten verfügbar werden, des Verifizierungsprozesses als ausreichend betrachtet.“

Trotz mangelnder zuverlässiger Referenzdaten sollen hier auch noch die Ergebnisse der im Rahmen unserer Projektgruppe durchgeführten Probeevakuierung

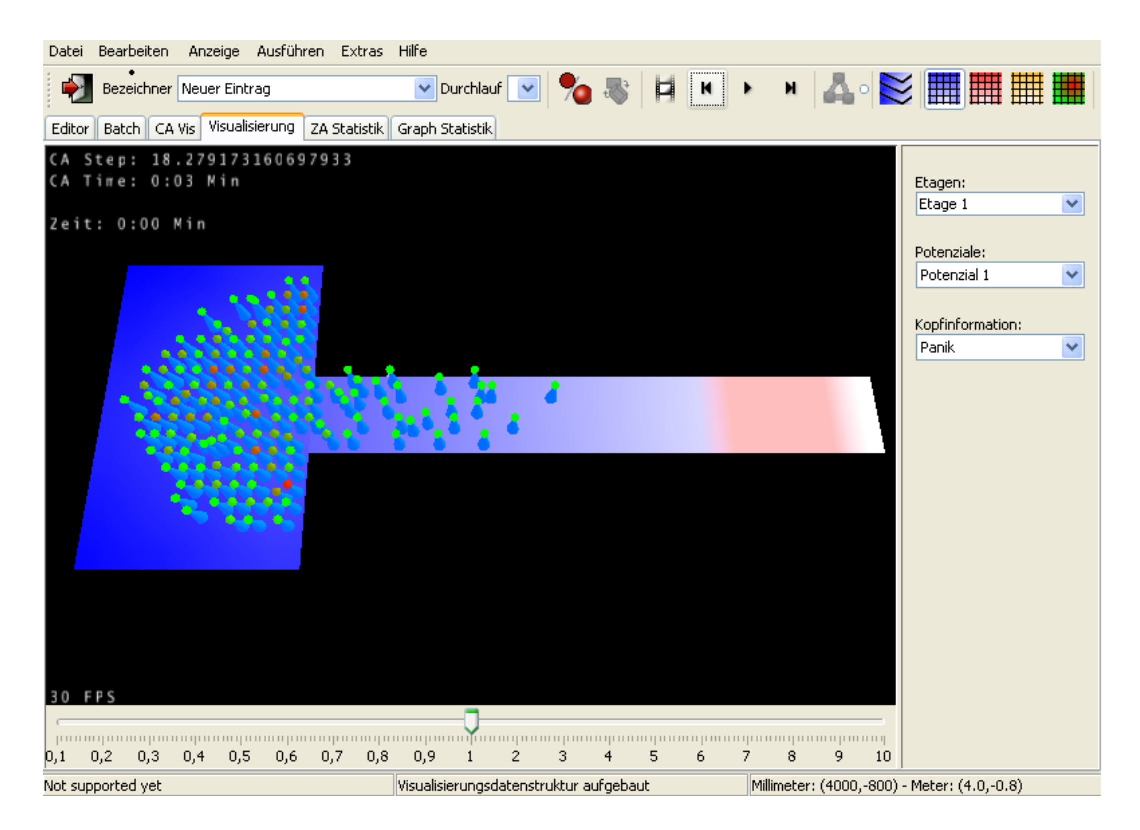



**Abbildung 8.37.:** Test 13 nach 3 Sekunden.

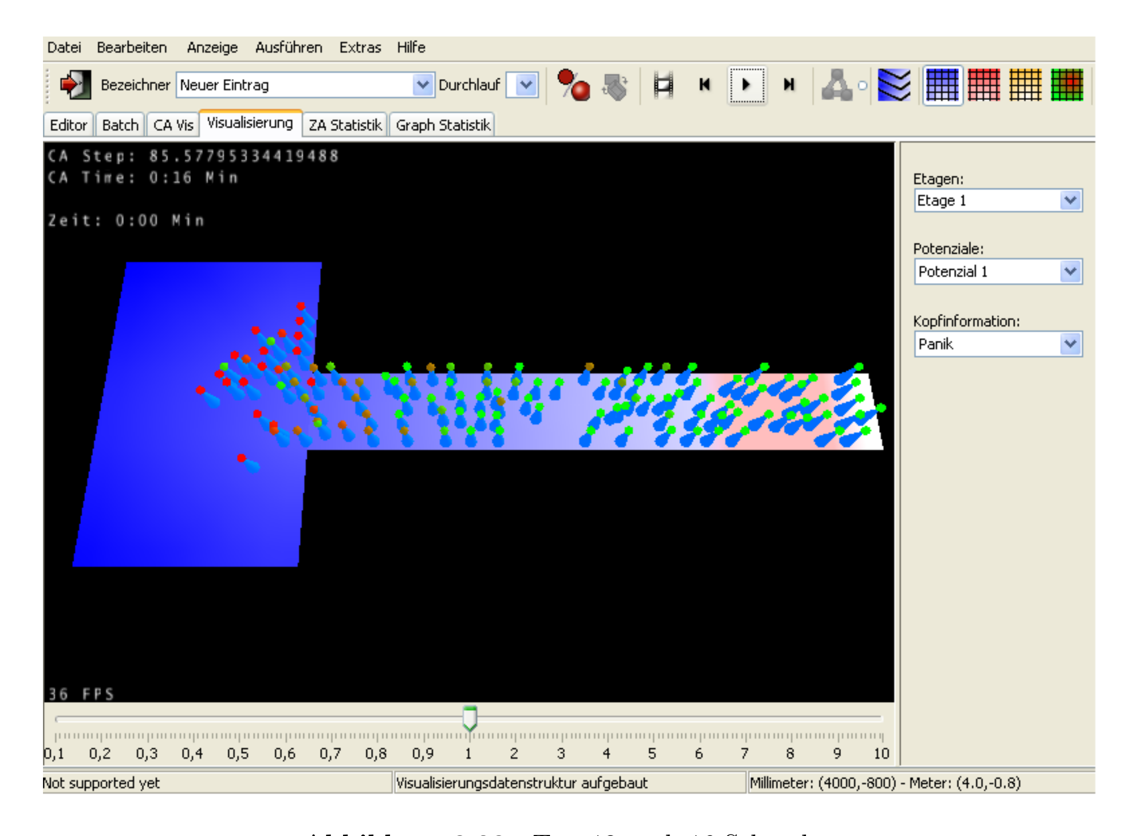



**Abbildung 8.38.:** Test 13 nach 16 Sekunden.

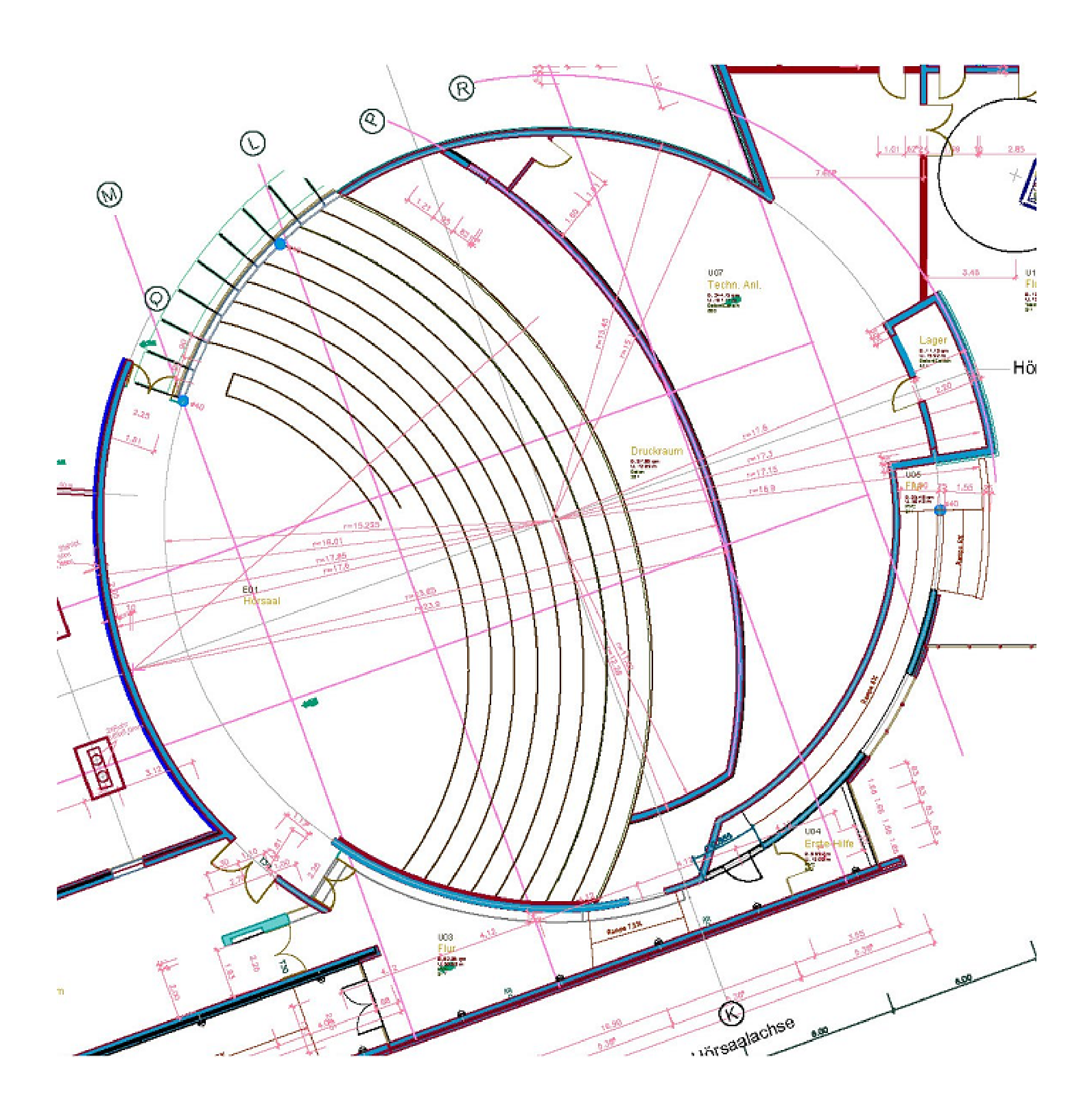



**Abbildung 8.39.:** Grundriss des Audimax.

des Audimax der Technischen Universität Dortmund vorgestellt werden. Die Evakuierung wurde gegen Ende einer Vorlesung durchgeführt, was einen relativ voll besetzten Hörsaal sicherstellte. Zum Zeitpunkt der Evakuierung befanden sich 325 Studenten im Audimax. Die Abbildung 8.39 zeigt den Grundriss des Audimax, die vorhandenen Ausgänge, sowie den in unserer Simulation verwendeten Modellplan (vgl. Abbildungen. 8.40, 8.41 und 8.42).

Nun sollen die Ergebnisdaten sowohl der tatsächlichen Probeevakuierung als auch der mit unserem Programm simulierten Evakuierung vorgestellt und verglichen werden. Die tatsächliche Evakuierung des Audimax dauerte 103 Sekunden. Unser Programm liefterte bei 10 Durchläufen eine durchschnittliche

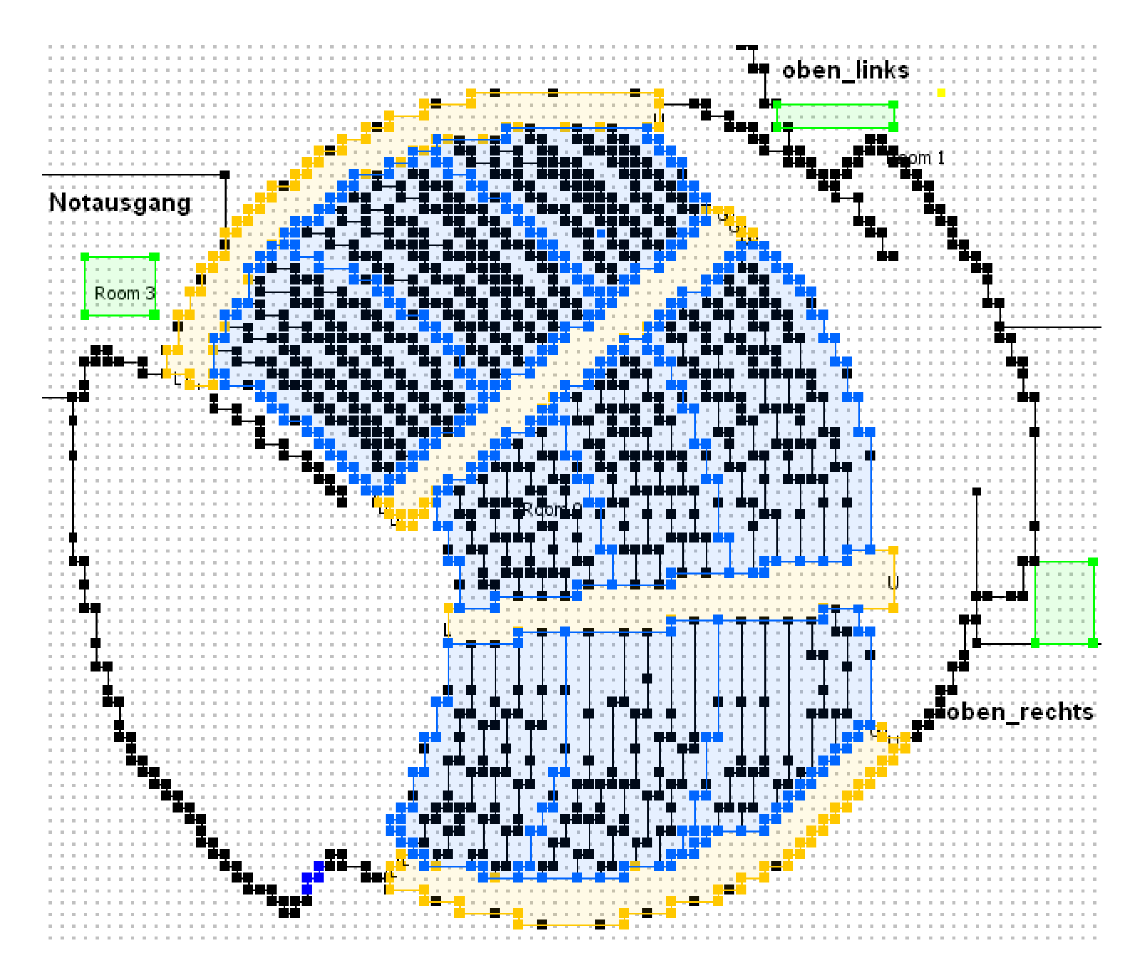



**Abbildung 8.40.:** Modell des Audimax in unserem Editor. (Der vierte Ausgang befindet sich auf der unteren Etage und ist hier nicht sichtbar)

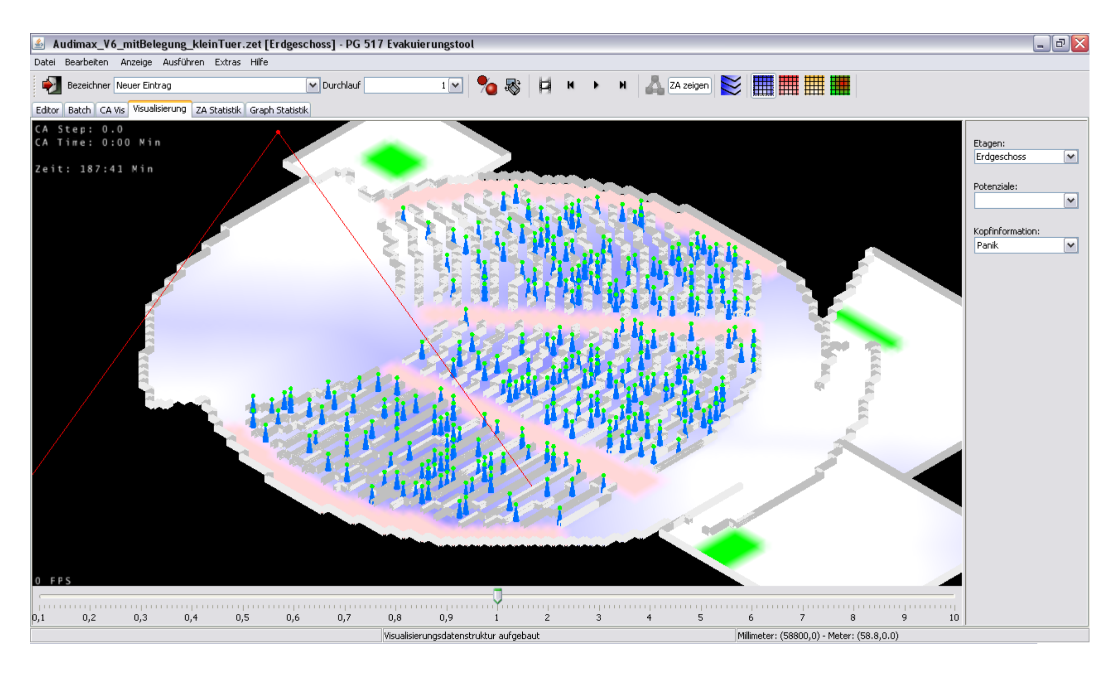



**Abbildung 8.41.:** Modell des Audimax in unserer Visualisierung in 3D Ansicht.

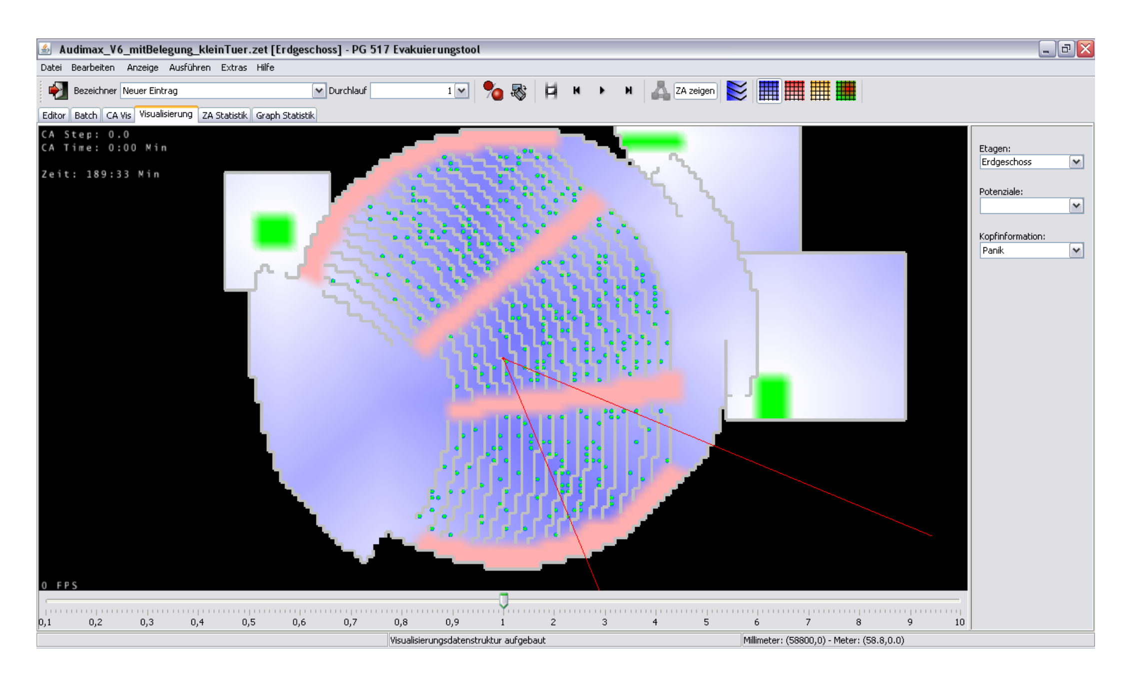



**Abbildung 8.42.:** Modell des Audimax in unserer Visualisierung mit Blick von oben.

Evakuierungszeit von 55 Sekunden. Die minimale Zeit betrug 44 Sekunden, die maximale 81 Sekunden. Die Diskrepanz zwischen realen Wert und simulierten Werten kann man sich sicherlich zum Teil darauf zurückführen, dass der reale Wert aus einer für alle Teilnehmer ersichtlichen Probeevakuierung stammt. Beim Verhalten der Personen war zu beobachten, dass diese sich ruhig und geordnet verhalten haben. In einer Notsituation würden sich die Personen sicherlich schneller bewegen und auch dichter gedrängt das Audimax verlassen.

Außerdem haben wir die Verteilung der Personen auf die Ausgänge untersucht. Das Ergebnis ist in den Abbildungen 8.43 und 8.44 zu sehen. Dabei fällt auf, dass der Ausgang unten rechts in der Realität wesentlich weniger genutzt wurde. Dies liegt wahrscheinlich daran, dass der Weg zur Mensa von dort aus weiter ist. Solche Faktoren berücksichtigt unser Programm natürlich nicht so genau. Es ist zwar möglich, den Attraktivitätswert der Ausgänge anzugeben, aber damit genau die richtigen Werte zu finden, ist schwierig. Weiterhin fällt auf, dass der Notausgang in unserem Programm weniger genutzt wird. Auch dies liegt an den Attraktivitätswerten der Ausgänge.

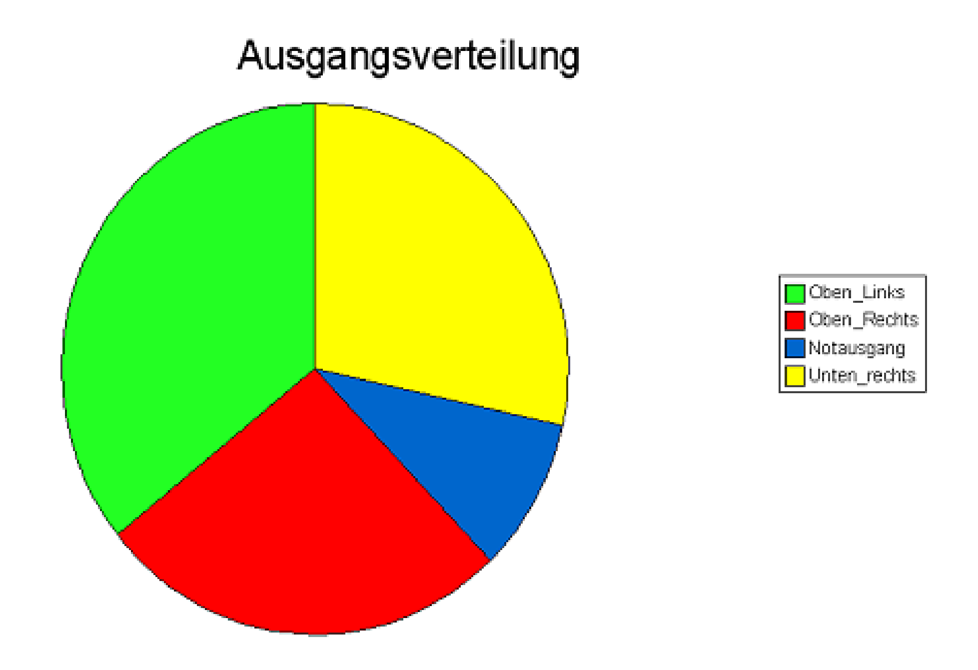



**Abbildung 8.43.:** Ausgangsverteilung der tatsächlichen Testevakuierung.

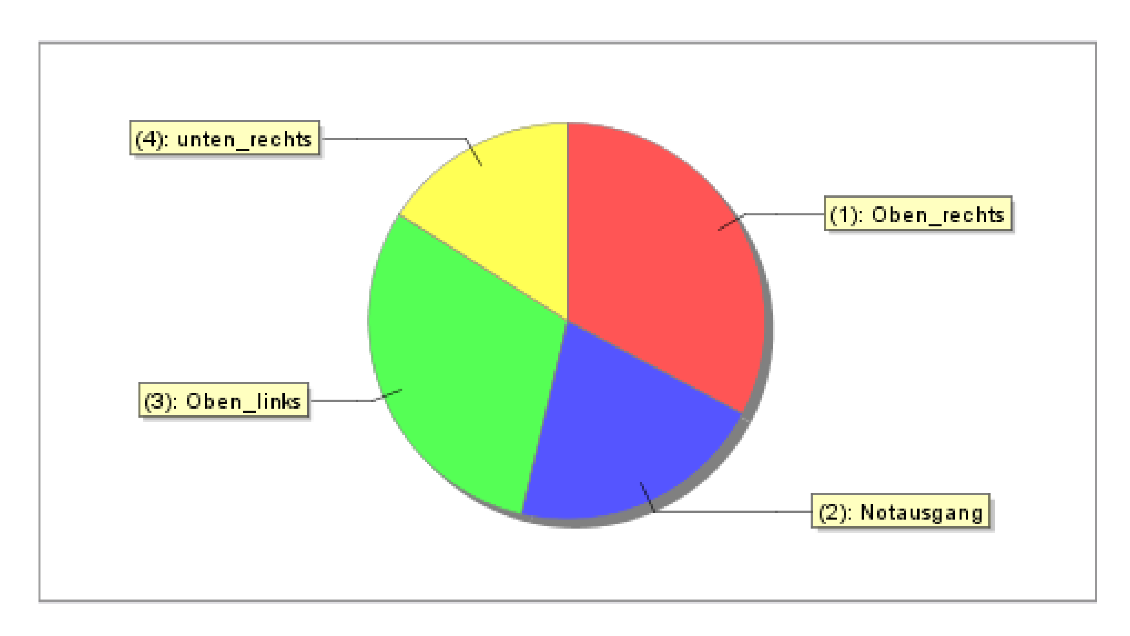



**Abbildung 8.44.:** Ausgangsverteilung der Evakuierung mit unserem Programm.

# Teil IV.

# Benutzerhandbuch

# Inhaltsverzeichnis

# 9. Einen Gebäudeplan erstellen

Bevor Sie mit dem Evakuierungstool eine Evakuierung durchführen können, müssen Sie im Z-Editor einen Gebäudeplan erstellen oder laden. Der folgende Abschnitt soll Ihnen dabei helfen. Er beschreibt den Aufbau der Benutzeroberfläche des Z-Editors und beschreibt beispielhaft die Erstellung eines Bauplans.

## 9.1. Allgemeine Programmoberfläche

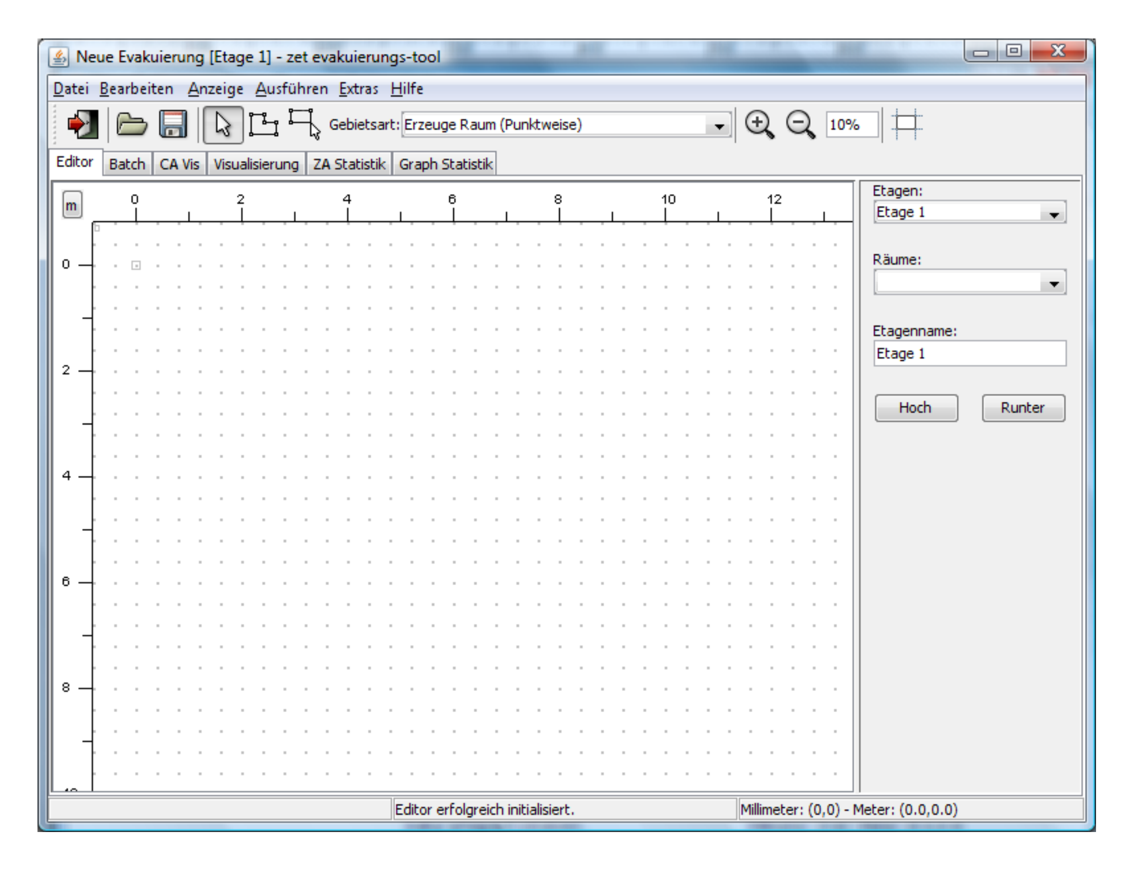



**Abbildung 9.1.:** Der Z-Editor direkt nach dem Programmstart mit einem neuen Projekt.

Die Programmoberfläche von ZET unterstützt verschiedene Modi, die über eine Reihe von Tabs im oberen Bereich des Programmfensters gewechselt werden können. Der mit EDITOR bezeichnete Tab symbolisiert den Modus zum Zeichnen von Plänen, dem Z-Editor, der nach dem Programmstart automatisch aktiv ist. Mit diesem Modus beschäftigen wir uns in diesem Abschnitt.

Über der Tableiste befindet sich eine Symbolleiste, in der verschiedene zum gewählten Modus passende Aktionen ausgewählt werden können. Die Leiste passt sich beim Wechsel des Modus automatisch an.

Am unteren Rand des Fensters befindet sich eine dreigeteilte Statusleiste.

## 9.2. Die Oberfläche des Z-Editors

Der Z-Editor stellt die Schnittstelle zwischen dem Benutzer und dem Z-Format dar. Mit ihm können neue Gebäudepläne und zugehörige Belegungen erzeugt und bearbeitet werden. Falls bereits andere Gebäudepläne existieren, kann der Editor den Benutzer bei der Übernahme dieser Pläne in das Z-Format unterstützen, indem der bestehende Plan im Hintergrund des Editors eingeblendet wird. Weiterhin gibt es einen speziellen Belegungseditor mit dem sowohl neue Belegungstypen als auch Personengruppen in bestehenden Typen angelegt und bearbeitet werden können. Informationen zum Belegungseditor befinden sich in Abschnitt 9.4.

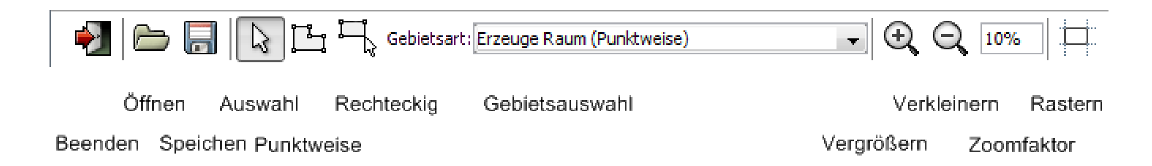



**Abbildung 9.2.:** Symbolleiste des Editier-Modus.

Die Symbolleiste des Zeichenmodus ist in Abbildung 9.2 dargestellt, die Funktionen werden in den entsprechenden Abschnitten erläutert. Die dreigeteilte Statusleiste zeigt im linken Bereich Fehler an, die der Arbeit auftreten. Im mittleren Bereich wird der aktuelle Status des Programms bzw. die als nächstes auszuführende Aktion ausgegeben, wie zum Beispiel „wählen Sie die Koordinaten“ nachdem der Button zum Zeichnen eines Raumes angewählt worden ist.

Im rechten Feld wird die aktuelle Position des Zeichenstiftes vermerkt, sowohl in Millimetern als auch in Metern.

### 9.2.1. Die Zeichenfläche

Im Zeichenmodus besteht der Rest des Fensters im Wesentlichen aus einer Zeichenfläche, auf der jeweils ein Stockwerk dargestellt wird. Standardmäßig kann nur in einem bestimmten Raster gezeichnet werden, die Punkte des Rasters werden durch graue Punkte dargestellt und liegen im Abstand von 40 cm. Dies ist die Einteilung, die vom zellulären Automaten und den Konvertierungsalgorithmen benötigt wird. Über ANSICHT | ZEICHNEN AUF RASTER kann die Einschränkung, nur auf Rasterpunkte zu zeichnen, ausgeschaltet werden. Das Häkchen vor dem Menüpunkt signalisiert den aktuellen Status. Mit den Untermenüpunkten von ANSICHT | GITTERNETZ können die grauen Punkte ausgeblendet oder durch ein Gitternetz ersetzt werden.

Die aktuelle Position des Zeichenstiftes wird durch ein kleines Viereck markiert. Es ist zu beachten, dass die Position im Rastermodus nicht an der Spitze des Mauszeigers liegt. Es wird jeweils der am nächsten zur Mausposition liegende Rasterpunkt ausgewählt. Die Koordinaten des Punktes werden in der Statuszeile angegeben.

Falls ohne Beschränkung auf Rasterpunkte gezeichnet wurde, kann der Plan über BEARBEITEN | RASTERN oder über die Schaltfläche RASTERN in der Menüleiste gerastert werden, das heißt alle Punkte werden auf naheliegende Rasterpunkte verschoben und schräge Linien werden durch treppenförmige, auf dem Raster liegende Linienzüge ersetzt. Dabei können vereinzelt Probleme auftreten, die nicht automatisch behoben werden können, in diesem Fall müssen nachträglich Verbesserungen am Plan vorgenommen werden.

Rechts neben der Zeichenfläche befindet sich ein in der Größe veränderbarer Eigenschaftenbereich. Dort werden Eigenschaften für das gerade markiertes Objekt angezeigt. Als erstes befinden sich im Panel immer zwei Auswahlfelder: Mit dem ersten kann die aktuelle Etage ausgewählt werden, mit dem zweiten wählt man ein Raum auf der Etage. Falls ein Raum ausgewählt wird, wird er markiert und, falls er außerhalb des sichtbaren Bereichs lag, der Plan so verschoben, dass der Raum sichtbar ist.

Falls kein Objekt markiert ist, werden die Eigenschaften für die aktuelle Etage angezeigt, dabei handelt es sich um den Namen der Etage, der auf diese Weise auch geändert werden kann. Mit den Schaltflächen Hoch und Runter kann die Etage nach oben und unten verschoben werden. Alternativ können die Menüeinträge Bearbeiten | Stockwerk nach oben und Stockwerk nach unten benuzt werden. Mit den Menüpunkten Bearbeiten | Neues Stockwerk und Bearbeiten | Stockwerk löschen können neue Stockwerke eingefügt oder gelöscht werden.

An der oberen und linken Seite des Zeichenbereichs befinden sich Lineale mit denen die Positionen der Räume ausgehend von einem Nullpunkt bestimmt werden können. In der Ecke, in der sich die Lineale kreuzen würden, befindet sich eine Schaltfläche, mit der die Einheit des Lineals verändert werden kann. Zur Verfügung stehende Einheiten sind Zentimeter, Dezimeter, Meter, Inch, Feet und Yard.

Mit dem Mausrad kann in den Plan hinein- und herausgezoomt werden, die Schaltflächen Plus und Minus verdoppeln bzw. halbieren den jeweils aktuellen Zoomfaktor, der auch in das nebenliegende Textfeld direkt eingegeben werden kann.

Objekte auf der Zeichenfläche können mit einem Doppelklick markiert werden. Falls sich mehrere Objekte an derselben Stelle befinden, können die Markierungen durch weitere Doppelklicks gewechselt werden. Im Eigenschaftenbereich rechts werden die für das markierte Objekt gültigen Eigenschaften angezeigt. Durch Ziehen mit dem Mauszeiger im Markierungsmodus (das ist der bei Programmstart eingestellte Modus) wird ein Markierungsrechteck angezeigt. Wird die Maustaste losgelassen, werden alle komplett innerhalb des Markierungsrechtecks befindlichen Objekte markiert, sie können dann auch alle gleichzeitig verschoben werden. Jedes Objekt kann durch Drücken von Entfernen gelöscht werden. Wird ein Raum auf einen Punkt geschrumpft, kann er nicht mehr mit einem Klick markiert werden; er kann jedoch mit dem Markierungsbereich angewählt und anschließend gelöscht werden.

## 9.3. Einen Plan zeichnen

Das Zeichnen eines neuen Plans, auf dessen Basis eine Evakuierungs-Simulation durchgeführt werden soll, wird im Folgenden erläutert. Dies soll möglichst praxisnah geschehen. Aus diesem Grund wird neben der abstrakten Beschreibung auch das Erstellen eines Plans für ein kleines Beispiel-Gebäude gezeigt. Der Bauplan für das Erdgeschoss dieses Gebäudes ist in Abbildung 9.3 zu sehen, der Bauplan für das 1. Obergeschoss in Abbildung 9.4.

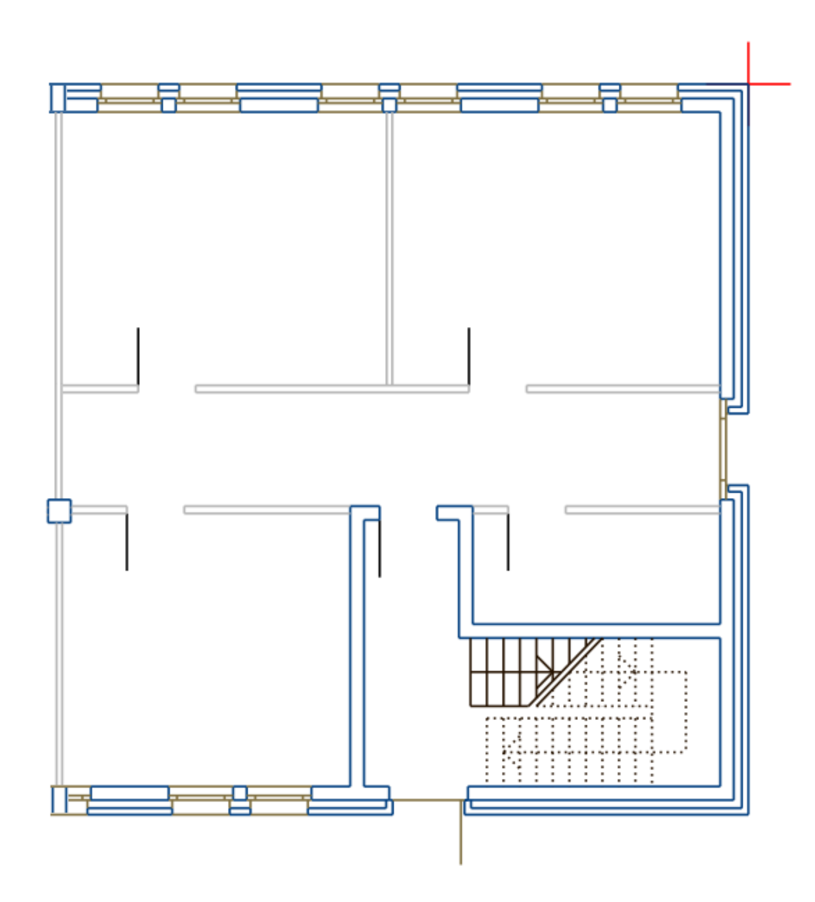


**Abbildung 9.3.:** Erdgeschoss des Beispiel-Gebäudes

### 9.3.1. Einen neuen Plan anlegen

Direkt nach dem Start ist die Zeichenfläche leer, der neue Plan enthält eine Etage. Über DATEI | NEU kann dieser Zustand wiederhergestellt werden; damit der alte Plan nicht versehentlich gelöscht wird, fragt das Programm nach, ob dieser gespeichert werden soll.

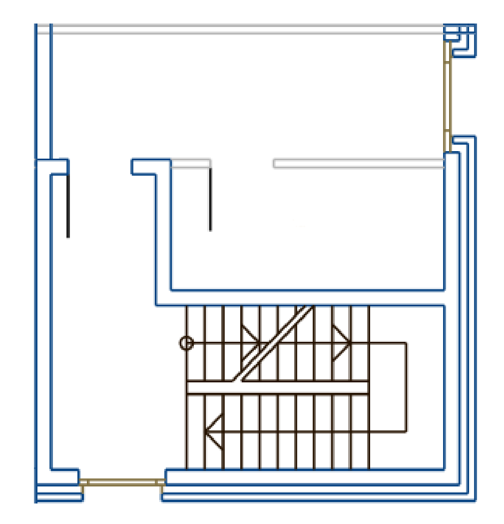


**Abbildung 9.4.:** 1. OG des Beispiel-Gebäudes

Pläne können über die Menüpunkte DATEI | SPEICHERN und DATEI | SPEICHERN UNTER... auf der Festplatte gespeichert werden. Falls ein neuer Plan noch nicht gespeichert worden ist, wird in einem Dialogfeld nach dem Dateinamen gefragt. Der zweite Menüpunkt ruft immer das Dialogfenster auf, so dass ein Plan unter einem neuen Namen erneut gespeichert werden kann. Mit dem Menüeintrag DATEI | ÖFFNEN... kann ein vorhandener Plan geladen werden. Alternativ können die Icons SPEICHERN und ÖFFNEN der Symbolleiste benutzt werden.

### 9.3.2. Räume und Gebiete zeichnen

Standardmäßig ist das Programm im Selektionsmodus, zu dem die Schaltfläche AUSWAHL wieder zurückführt. In diesem Modus können existierende Räume markiert werden. Zum Zeichnen muss einer der Modi PUNKTWEISE ZEICHNEN oder RECHTECKIG ZEICHNEN ausgewählt werden. Das zum aktuellen Modus gehörende Icon wird hervorgehoben.

Im Modus zum punktweisen Zeichnen erzeugt jeder Mausklick in den Editierbereich einen Punkt an der Stelle, an der sich der Zeichenstift befindet. Aufeinanderfolgende Punkte werden durch eine gerade Linie verbunden, die neu hinzugefügte Linie folgt dem Mauszeiger. Durch Drücken der rechten Maustaste wird der letzte Punkt mit dem Ersten berbunden, so dass ein geschlossenes

Polygon entsteht, das die inneren Wände eines Raumes darstellt. Ein gültiger Raum darf sich nicht selbst schneiden.

Im Rechteckmodus wird durch den ersten Klick auf die Zeichenfläche eine Ecke eines rechteckigen Raumes bezeichnet, der zweite Klick bestimmt die diagonal gegenüberliegende Ecke. Der Umriss wird ebenfalls angezeigt und folgt bei Mausbewegungen dem Zeichenstift.

Hat man schon mit dem Zeichnen begonnen und möchte es wieder abbrechen, kann man dafür die Escape-Taste drücken.

Neu gezeichnete Räume erhalten einen Namen mit fortlaufender Nummerierung. Wenn ein Raum markiert wird, kann sein Name im Eigenschaftenbereich verändert werden. Wenn der Name in den Raum hineinpasst, wird er im Zentrum der konvexen Hülle des Polygons angezeigt, dieses kann bei nicht konvexen Räumen auch außerhalb des Raumes liegen.

Auf die gleiche Art können alle Gebiete (Unbetretbare Bereiche, Barrieren sowie Belegungs-, Sicherheits-, Evakuierungs-, Verzögerungs- und Treppenbereiche) erzeugt werden. Neben den Schaltflächen zur Auswahl des Zeichenmodus befindet sich dazu die Gebietsauswahl (Zu beachten ist, dass Barrieren nur im Modus „punktweise Zeichnen" zur Verfügung stehen). Ein Bereich liegt immer in einem bestimmten Raum, der sich nicht ändern kann. Dieser Raum ist derjenige, in den zuerst hineingeklickt wird. Alle weiteren Eckpunkte müssen dann auch in diesem Raum liegen, andernfalls blinkt im Fehlerbereich der Statusleiste eine entsprechende Nachricht. Wird ein bestehendes Gebiet aus dem Raum, zu dem es gehört, hinausgeschoben, wird es entfernt.

Wenn mit der Gebietsauswahl eine Gebietsart ausgewählt wird, wird automatisch in einen Zeichenmodus gewechselt. Ob der punktweise Modus oder der Rechteckmodus gewählt wird, hängt vom vorher zuletzt benutzen Modus ab.

Bestehende Räume können nachträglich bearbeitet werden. Dazu muss ein Raum zunächst markiert werden, anschließend können einzelne Wände und auch Eckpunkte mit der Maus verschoben werden. Die anderen Wände passen ihre Länge und Lage automatisch an die neuen Positionen an.

Über Anzeige | Sichtbare Bereiche können die einzelnen Bereiche ausgeblendet und wieder eingeblendet werden.

Zusätzlich haben Punkte und Wände Kontextmenüs, die über einen Klick mit der rechten Maustaste geöffnet werden können. Über PUNKT LÖSCHEN im Kontextmenü eines Punktes kann er gelöscht werden. Über NEUEN PUNKT EINFÜGEN im Kontextmenü von Kanten kann an der entsprechenden Stelle ein Punkt eingefügt werden. Dabei muss beachtet werden, dass es unter Umständen schwer ist, eine nicht waagerecht oder senkrecht verlaufende Linie zu treffen.

Wir wollen nun die verschiedenen Gebiete erläutern. Verzögerungsbereiche werden standardmäßig rot gezeichnet. Im Eigenschaftenbereich können der Verzögerungstyp und ein Verzögerungsfaktor eingestellt werden. Es sind nur die beiden Typen HINDERNIS und SONSTIGES hinterlegt. Zu jedem Hindernistyp gehört ein standardmäßiger Verzögerungsfaktor, für die beiden Typen sind diese $0,6$ und $1$. Durch Auswahl der Schaltfläche STANDARDWERT SETZEN wird der für den gewählten Typ gültige Faktor gewählt. Bei der Eingabe eines neuen Faktors muss die Eingabe durch Drücken von ENTER bestätigt werden.

Treppenbereiche werden orange dargestellt und repräsentieren eine Treppe. Nach dem Zeichnen des Gebietes müssen das untere und obere Ende der Treppe markiert werden, die Texte WÄHLEN SIE DAS UNTERE TREPPENENDE und WÄHLEN SIE DAS OBERE TREPPENENDE im Statusbereich deuten darauf hin. Die Kantenauswahl geschieht durch Klicken mit der linken Maustaste auf eine Kante. Die bezeichneten Kanten werden mit L und U (für lower und upper) markiert. Das erfolgreiche Erzeugen einer Treppe wird durch den Statustext TREPPE ERFOLGREICH ERZEUGT! gemeldet. Treppen haben zwei Verzögerungsfaktoren, die für die Benutzung nach oben und unten benutzt werden. Auch die Eingabe dieser Faktoren muss mit ENTER bestätigt werden.

Sicherheitsbereiche werden gelb dargestellt und haben keine Eigenschaften, die eingestellt werden können. Wenn Personen einen Sicherheitsbereich betreten, gelten sie als evakuiert, sie befinden sich jedoch noch in der Simulation (und können so zum Beispiel Staus hinter Ausgangstüren erzeugen). Wenn von einem Sicherheitsbereich ein Evakuierungsbereich erreichbar ist (ohne dabei den sicheren Bereich zu verlassen), versuchen gesicherte Personen diesen zu erreichen. Falls mehrere Evakuierungsbereiche erreichbar sind, wird der am nächsten liegende ausgewählt.

Evakuierungsbereiche werden grün gezeichnet, haben eine Attraktivität und

einen Namen. Personen versuchen bei einer Evakuierung einen Evakuierungsbereich zu erreichen; wenn sie diesen erreicht haben, sind sie endgültig sicher und direkt evakuiert. Im Graphen fungieren die zu diesen Bereichen gehörenden Knoten als Senken, in der Simulation gehen von diesen Zellen Potentiale aus. Der Name wird in der Statistik benutzt um die Ausgänge zu identifizieren. In einigen Regeln wechseln Personen ihr Potential, dies kann zum Beispiel nach der Attraktivität geschehen. Höhere Werte sind dabei attraktiver. So kann zum Beispiel der Haupteingang eines Gebäudes als besonders attraktiv gekennzeichnet werden, so dass er vorwiegend benutzt wird, obwohl Notausgänge näher liegen.

Belegungsbereiche definieren einen Bereich, in dem sich während der Berechnungen Personen aufhalten. In der Editor-Ansicht werden sie standardmäßig blau dargestellt. Damit in ein Gebiet nicht zu viele Personen eingetragen werden, wird im Eigenschaftenbereich die Größe des Bereichs und die maximale Anzahl Personen, die dort hineinpassen, angezeigt. Im Feld PERSONEN kann die Personenzahl, die sich im Bereich befinden, angegeben werden. Drücken der Schaltfläche STANDARDPERSONENZAHL legt die Personenzahl auf die für den gewählten Belegungstyp geltende Standardpersonenzahl fest, falls soviele überhaupt hineinpassen. Ansonsten wird der kleinere Wert gewählt. Auch wenn ein Belegungsbereich gezeichnet wird, werden soviele Personen wie im Standard vorgegeben, oder wie maximal hereinpassen, eingetragen. Falls manuell zu viele Personen gesetzt werden, oder falls durch spätere Formänderungen die Personenzahl verringert worden ist, wird beim Aufruf der Simulation eine Fehlermeldung ausgegeben. Den Belegungstyp der für ein Belegungsgebiet gelten soll, kann ebenfalls über eine Listenauswahl gewählt werden. Es sind dort nur Einträge die zur aktuell aktiven Belegung zählen anwählbar. Genauere Informationen zum Verwalten von Belegungen befinden sich in Abschnitt 9.4.

Für Belegungsbereiche existiert eine spezielle Methode, mit der sie sich schneller erzeugen lassen. Bei Rechtsklick in einen Raum öffnet sich das Kontextmenü STANDARDBELEGUNG unter dem sämtliche gerade wählbaren Belegungstypen aufgeführt sind. Wählt man einen dieser Typen aus, wird der gesamte Raum mit einem Belegungsgebiet ausgefüllt und die Standardpersonenzahl vergeben, falls möglich.

### 9.3.3. Räume mit Türen verbinden

Wenn zwei Kanten genau gleich lang sind und übereinander liegen, können sie zu einem Durchgang verbunden werden. Dazu muss im Kontextmenü einer der übereinanderliegenden Kanten Durchgang erzeugen gewählt werden. Die Kante wird anschließend gestrichelt gezeichnet und Individuen können hindurchgehen.

Alternativ können zwei existierende Kanten mit Hilfe eines kleinen Raumes verbunden werden. Dazu muss nacheinander bei beiden Kanten im Kontextmenü Zwei Kanten mit Tür verbinden ausgewählt werden. Die Eckpunkte der Kanten werden durch weitere Kanten verbunden, so dass ein neuer Raum entsteht, und die vorhandenen Kanten werden passierbar. Die Lage und Größe der Kanten ist dabei irrelevant.

Genauso wie normale Durchgänge können auch Räume auf verschiedenen Etagen verbunden werden. Dazu müssen die Kanten jedoch gleich lang sein und entweder beide horizontal oder vertikal verlaufen. Nachdem im Kontextmenü der ersten Kante Stockwerkübergang erzeugen ausgewählt wurde, muss das Stockwerk gewechselt werden und anschließend der gleiche Menüpunkt auf der anderen Kante gewählt werden. Die Kanten müssen nicht genau aufeinander liegen, allerdings dürfen sie nicht auf dem gleichen Stockwerk liegen. Kanten zwischen zwei Stockwerken werden gestrichelt und blau dargestellt.

Jede Kante kann auch mit dem Menüeintrag Evakuierungsausgang erzeugen in einen Ausgang aus dem Gebäude verwandelt werden. Das Nutzen von Evakuierungsausgängen erlaubt es, ohne speziell ausgezeichnete Evakuierungsbereiche auszukommen. Sobald ein Evakuierungsausgang erzeugt wird, wird automatisch ein Raum auf der **Evakuierungsetage** angelegt, welcher komplett aus einem Evakuierungsbereich besteht und über eine Tür mit der Kante verbunden ist. Da der Durchgang auf eine andere Etage führt, ist er ebenfalls blau dargestellt. Die **Evakuierungsetage** ist standardmäßig versteckt, kann jedoch über Anzeige | Standardevakuierungsbereich verstecken ein- und ausgeblendet werden.

Wird ein Durchgang rechts angeklickt, kann mit dem Befehl Durchgang entfernen des Kontextmenüs die Passierbarkeit zurückgenommen werden.

Wird ein Stockwerkübergang rechts angeklickt, gibt es zusätzlich den Menüeintrag Wohin führt diese Kante?. Bei Wahl dieses Menüpunktes wird der zugehörige Raum auf dem anderen Stockwerk markiert und im Bildschirm angezeigt. Dies funktioniert auch mit Evakuierungsausgänge, auch wenn die Standardevakuierungsetage ausgeblendet ist.

### 9.3.4. Praktisches Beispiel für das Erstellen eines Plans

Im nun folgenden Teil soll nun das angekündigte Beispiel für das Erstellen eines Plans für ein Gebäude erfolgen, dessen Baupläne vorliegen. In diesem Beispiel wird dabei von dem Gebäudeplan ausgegangen, der in den Abbildungen 9.3 und 9.4 zu sehen ist.

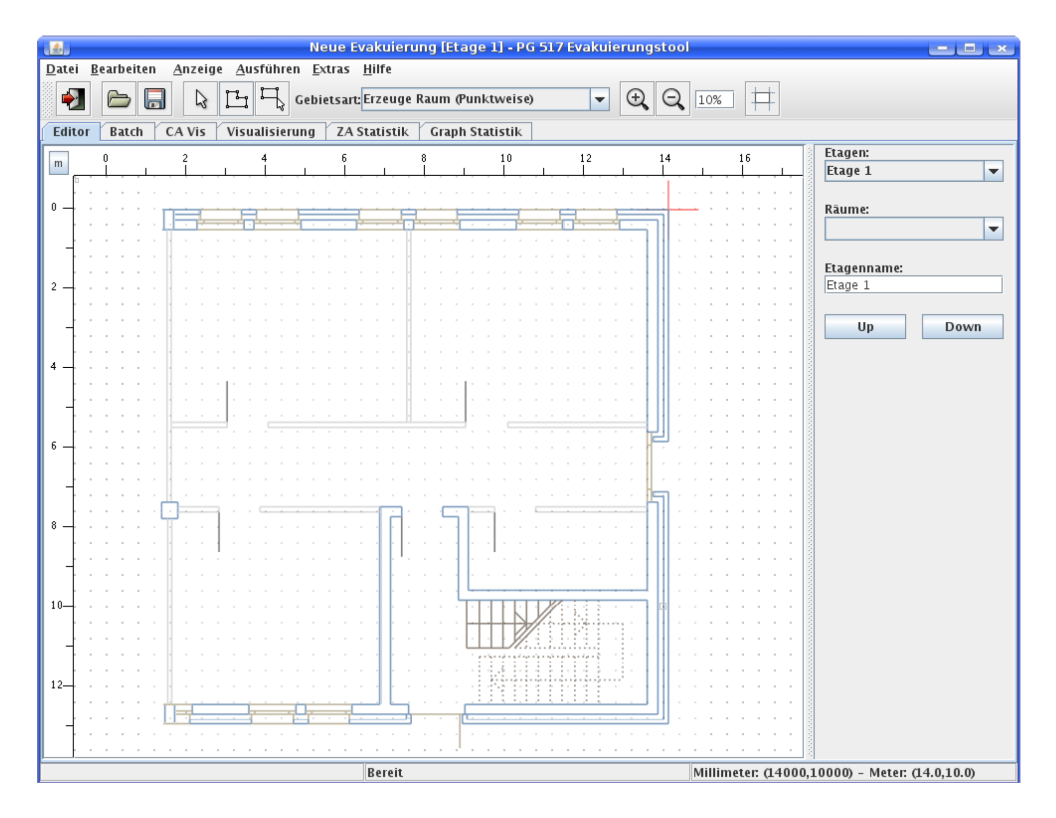



**Abbildung 9.5.:** Der Editor mit eingeblendetem Erdgeschossplan.

Um den Bauplan des Gebäudes möglichst komfortabel abzeichnen zu können, bietet es sich an, diesen im Editor einzublenden. Dies ist über die Funktion Extras | Bildplan | Einblenden möglich. Durch Wählen dieses Menüpunktes öffnet sich ein Datei-Öffnen-Dialog, der es erlaubt, den entsprechenden Bauplan, der im JPEG-, PNG- oder GIF-Format als Bild vorliegen muss,

zu laden.

Anschließend öffnet sich dann ein neues Fenster, in dem man spezifizieren muss, wieviele Pixel der Grafik wievielen Metern in der Realität entsprechen, damit der geladene Bauplan auch maßstabsgetreu auf der Zeichenfläche eingeblendet wird. In diesem Beispiel entsprechen 40 Pixel 1 Meter. Auf Wunsch lässt sich über den Menüpunkt EXTRAS | BILDPLAN | POSITION ANPASSEN noch die Position des eingeblendeten Plans auf der Zeichenfläche über die Angabe eines Offsets variieren. Für das hier verwendete Beispiel wurde ein $x$-Offset von 2 Metern verwendet, das Ergebnis in Abbildung 9.5 zu sehen.

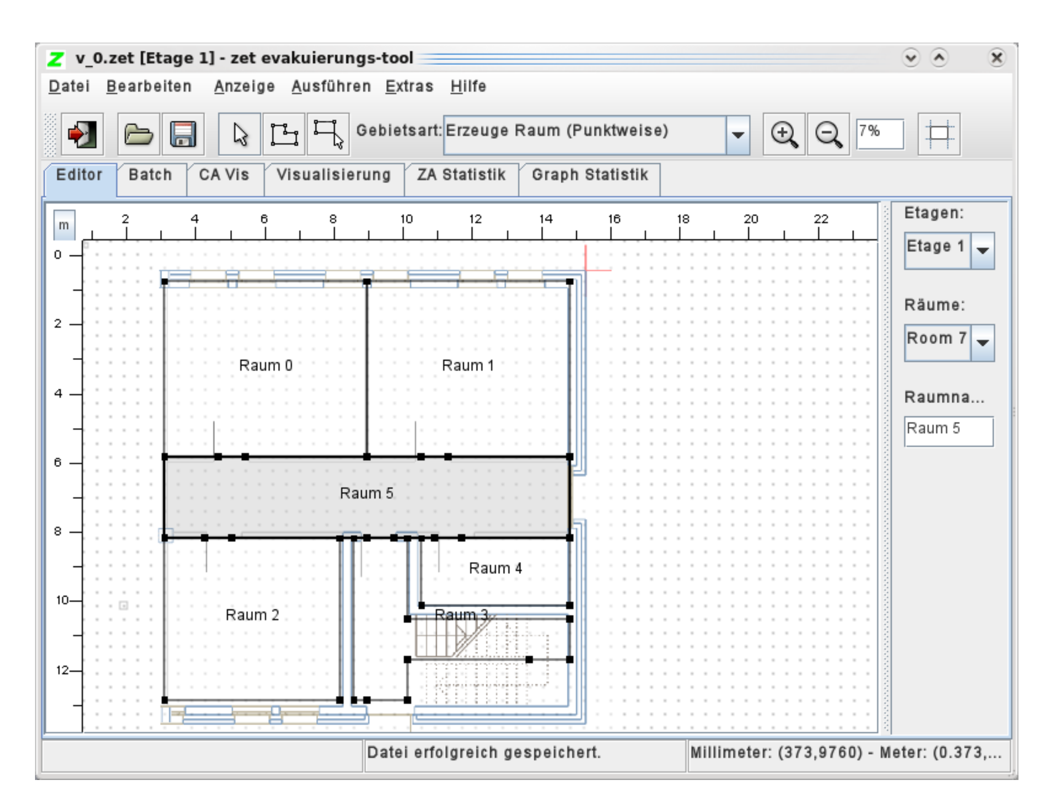



**Abbildung 9.6.:** Erdgeschossplan im Editor mit eingezeichneter Raumstruktur

Im Folgenden ist es dann zuerst notwendig, dass alle Räume des Bauplans auch auf der Zeichenfläche des Editors eingezeichnet werden. Dies geschieht, wie in der vorausgegangenen theoretischen Beschreibung erläutert, entweder über das Aufziehen eines Rechtecks oder über das Definieren einer geschlossenen Punktfolge, wenn unter GEBIETSART der Punkt ERZEUGE RAUM gewählt ist. Dabei ist aber zu beachten, dass nicht nur die Ecken des Raumes durch Punkte einer Punktfolge gekennzeichnet sein müssen, sondern ebenfalls alle weiteren Elemente der Gebäudestruktur, wie beispielsweise Türen, die Räume verbinden,

oder Ausgänge, die aus dem Gebäude herausführen. Es ist dabei wichtig zu unterscheiden, ob die *Innen*umrisse des Raumes oder die *Außenumrisse* des Raumes nachzeichnet werden: Wände werden in ZET nicht direkt gezeichnet, sie werden indirekt als Abstand zwischen den Innenumrissen der Räume dargestellt. Falls die Wände des Gebäudes erheblich dünner sind als 40 cm, sollte man sie allerdings weglassen und keinen Abstand zwischen den Räumen lassen. So ist es auch in unserem Beispiel geschehen. In Abbildung 9.6 ist hier die Phase des Plans zu sehen, in der die Gebäudestruktur auf die Zeichenfläche übertragen wurde. Dabei ist zu erkennen, dass die nach oben führende Treppe als eigenständiger Raum modelliert wurde, ebenso wie die hier in diesem Beispiel nicht existente nach unten führende Treppe. Dies mag auf den ersten Blick zwar seltsam erscheinen, hat aber einen durchaus wichtigen Hintergrund. Deshalb wird darauf noch später, wenn das Modellieren der Treppen genauer erläutert wird, eingegangen.

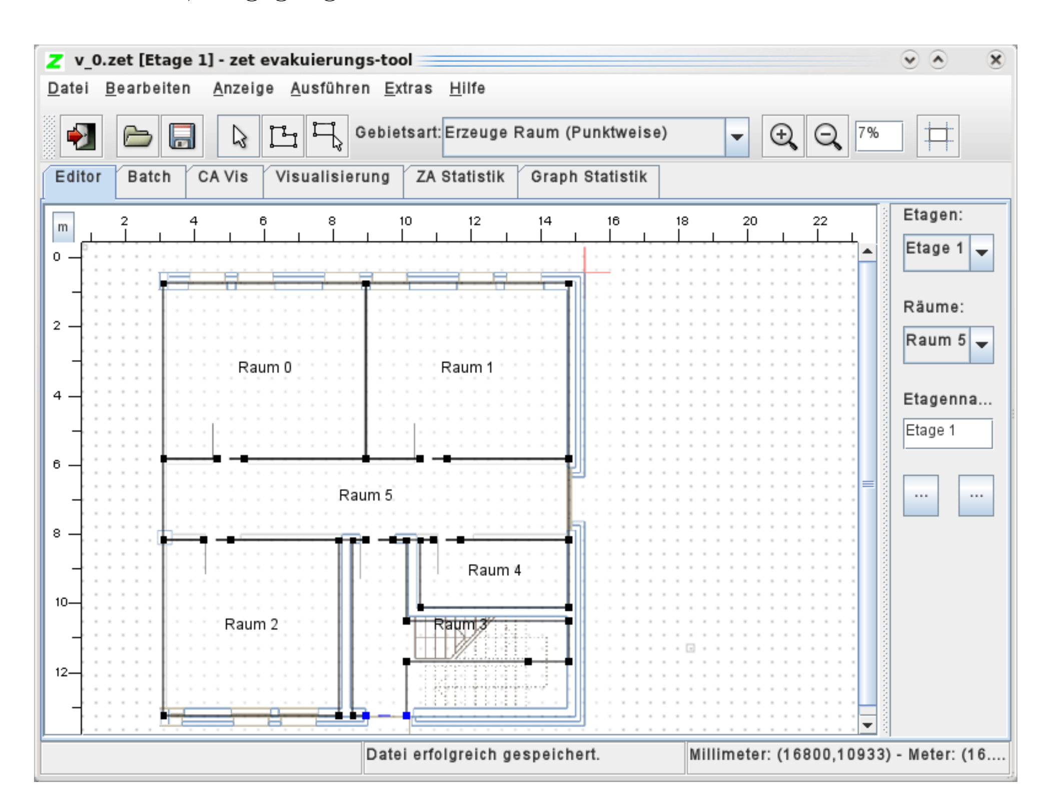



**Abbildung 9.7.:** Erdgeschossplan im Editor mit eingezeichneten Türen

Zu diesem Zeitpunkt sind noch alle Räume in diesem modellierten Plan von einander unabhängig, d.h. es gibt insbesondere auch noch keine Verbindungen zwischen den modellierten Räumen, über die es den flüchtenden Individuen

möglich wäre, von einem Raum in einen anderen zu gelangen. Deshalb macht es Sinn, im nächsten Schritt die Türen des Bauplans in den modellierten Plan zu übernehmen. Dazu klickt man mit der rechten Maustaste auf eine Tür, die im Modell einer Kante des Raumes entspricht, und wählt im daraufhin aufklappenden Menü den Punkt Durchgang erzeugen. Dabei ist es wichtig, zu wissen, dass das Erzeugen eines Durchgangs nur funktioniert, wenn die Kanten der beiden Räume, die zu einem Durchgang (bzw. einer Tür) verbunden werden sollen, exakt übereinander liegen. Falls das Erzeugen eines Durchgangs erfolgreich war, wird die gemeinsame Kante der beiden Räume, die von nun an den Durchgang repräsentiert, gestrichelt angezeigt, wie in Abbildung 9.7 zu erkennen ist. Der an der Südseite befindliche Ausgang des Gebäudes wird ebenfalls mit einem Rechtsklick auf die Tür-Kante und der Auswahl des Menüpunktes Evakuierungsausgang erzeugen angelegt. Natürlich müssen auch die Räume, die die Treppen repräsentieren, über Durchgänge angeschlossen werden.

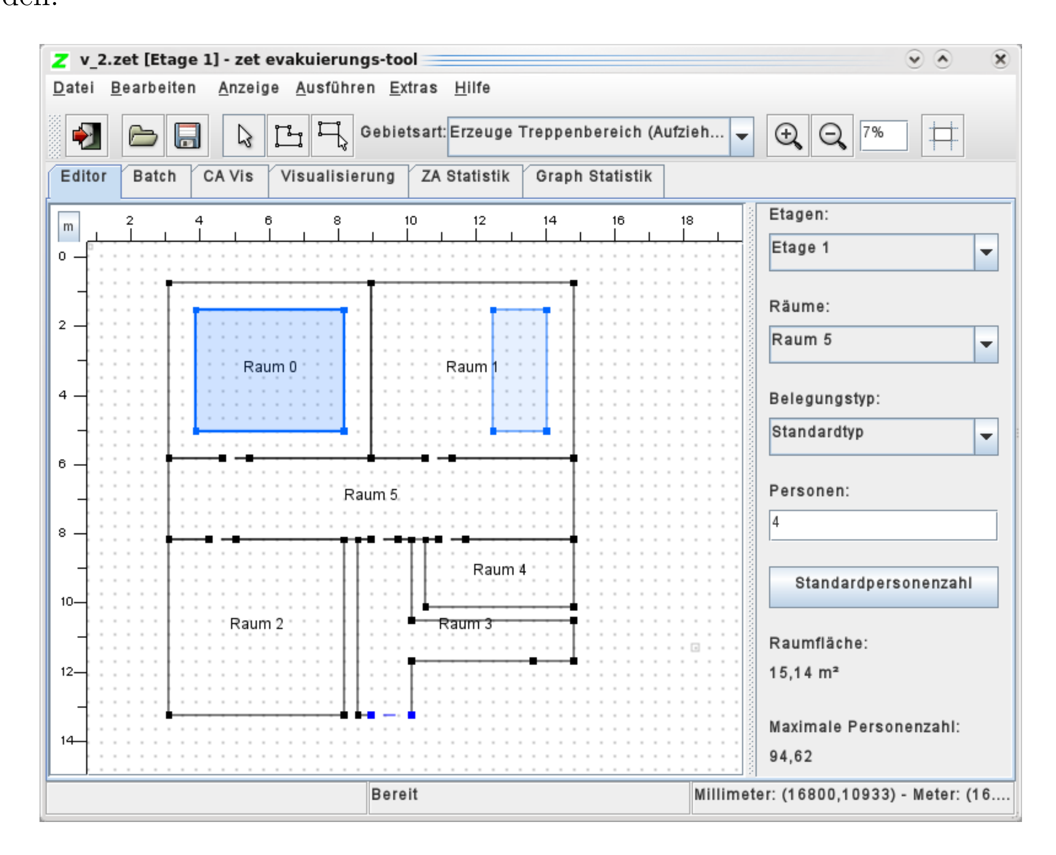



**Abbildung 9.8.:** Erdgeschossplan im Editor mit Belegungen

Als nächstes können die Räume jetzt je nach Bedarf mit Personenbelegungen versehen werden. Dies ist entweder über einen Rechtsklick auf einen Raum

und die Auswahl von Standard Belegung | Standardtyp (für den Fall, dass die Standard-Belegung gewünscht ist) möglich, oder alternativ, falls die gewünschte Belegung nicht den kompletten Raum ausfüllen soll, auch über das Zeichnen eines Rechtecks oder einer Folge von Punkten innerhalb eines Raums, wenn unter Gebietsart der Punkt Erzeuge Belegungsbereich gewählt ist. Ein Belegungsbereich ist im Editor blau gekennzeichnet, wie in Abbildung 9.8 erkennbar ist, in der in "Raum 0" und "Raum 1" jeweils ein Belegungsbereich eingezeichnet ist. Im Bereich rechts neben dem Zeichenfenster ist es nach Doppelklick auf einen Belegungsbereich möglich, die Anzahl der Personen, die sich in dem entsprechenden Belegungsbereich befinden sollen, zu konfigurieren. Dabei ist die maximale Anzahl an Personen, die sich in einem Belegungsbereich befinden können, abhängig von der Größe des Belegungsbereichs. Sie wird im Bereich rechts neben dem Zeichenfenster eingeblendet.

Ab dieser Stelle ist das Erdgeschoss des Gebäudes fertig modelliert (bis auf die Treppe) und die Konstruktion des Obergeschosses aus Abbildung 9.4 kann beginnen. Dazu muss über den Menüpunkt Bearbeiten | Neues Stockwerk ein weiteres Stockwerk erzeugt werden. Wenn man dieses nun im Bereich rechts neben der Zeichenfläche unter Etagen im Drop-Down-Menü wählt, erhält man für dieses neu erzeugte Stockwerk wieder eine leere Zeichenfläche. Auf dieser kann nun analog zum oben beschriebenen Vorgehen zum Zeichnen des Erdgeschosses das Obergeschoss modelliert werden. Der fertig modellierte Plan ist in Abbildung 9.9 zu sehen.

Nachdem beide Etagen des Gebäudes modelliert sind, müssen diese noch über eine Treppe miteinander verbunden werden, damit Personen zwischen den beiden Etagen wechseln können. Eine Treppe ist ein Gebiet, dass entweder als Rechteck oder geschlossene Punktfolge gezeichnet wird. Dazu muss unter unter Gebietsart die Auswahlmöglichkeit Erzeuge Treppenbereich gewählt werden. Nachdem der Bereich komplett gezeichnet ist, muss nacheinander das untere und obere Ende der Treppe markiert werden, da so unterschiedliche Geschwindigkeiten, abhängig von der Richtung, in der die Treppe benutzt wird, simuliert werden können. Zum Markieren müssen nacheinander zwei Kanten des Treppenbereiches mit der Maus angeklickt werden, ein entsprechender Hinweis wird in der Statuszeile angezeigt. Im Editor wird das untere Treppenende mit "L" (für lower) und das obere Treppenende mit "U" (für upper) bezeichnet.

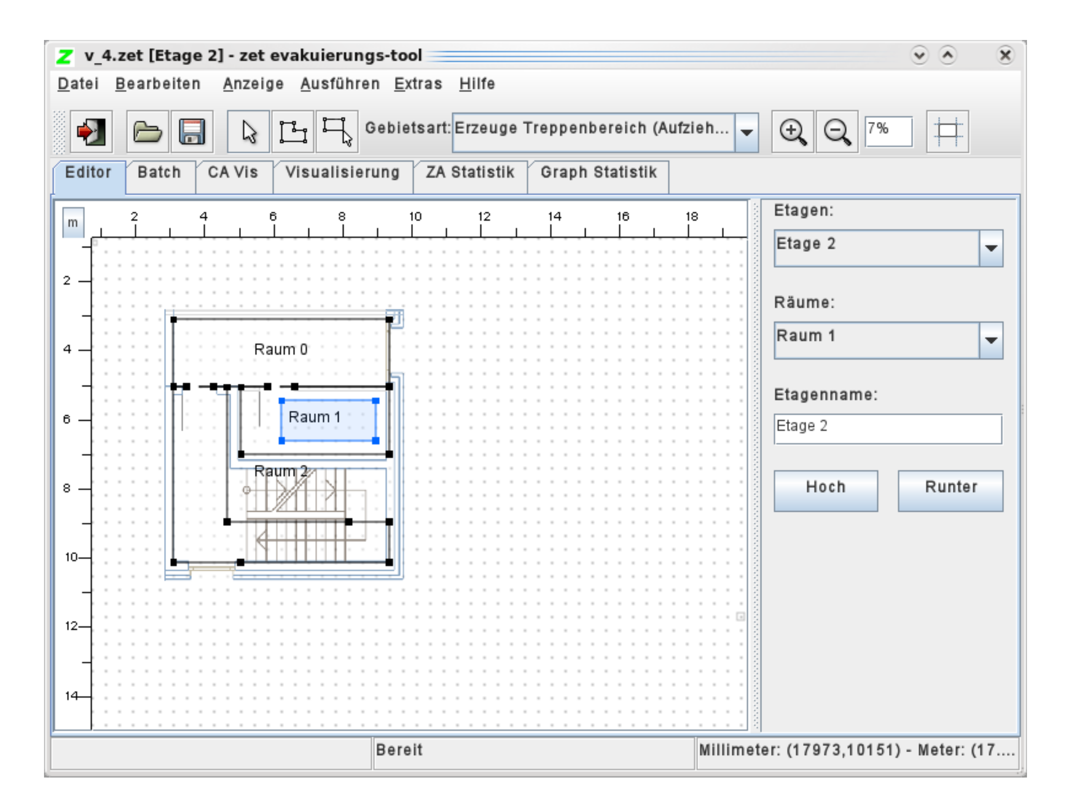



**Abbildung 9.9.:** Fertiger Plan des Obergeschosses

Da Treppenbereiche, wie im vorherigen Absatz bereits erwähnt, lediglich normale Bereiche innerhalb eines Raumes sind, ist es den flüchtenden Personen in der Simulation möglich, einen Treppenbereich prinzipiell an jeder beliebigen Stelle zu betreten und auch zu verlassen. Diese Eigenschaft eines Treppenbereichs entspricht natürlich nicht der Realität, in der Treppen üblicherweise nur am oberen oder unteren Ende betreten oder verlassen werden können. Um dies zu erreichen muss also die Struktur des Raumes, in dem sich die Treppe befindet, so angepasst werden, dass das Betreten bzw. Verlassen eines Treppenbereichs nur an den gewünschten Stellen möglich ist. Aus diesem Grund wurde der Plan des Gebäudes hier auch so konstruiert, dass für die Treppe ein eigenständiger (logischer) Raum modelliert wurde, der über einen Durchgang mit anliegenden Räumen verbunden werden kann. Dadurch ist sichergestellt, dass der Treppenbereich, nur über einen dafür vorgesehenen Durchgang zu einem Nachbarraum (und damit nicht mehr an jeder beliebigen Stelle) betreten oder verlassen werden kann. Eine andere, gleichwertige Möglichkeit, dieses Verhalten zu erwirken besteht darin, die Treppe mit Barrieren zu umgeben, so dass sie nur von einer Seite aus betreten werden kann.

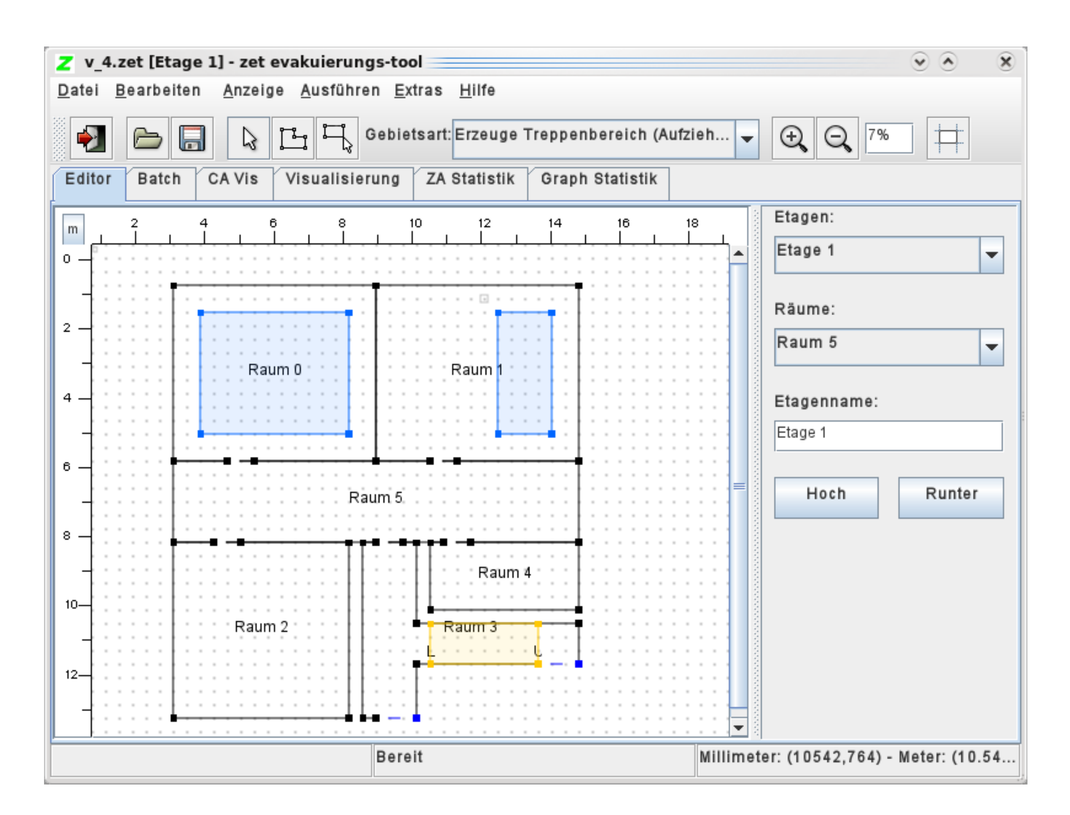



**Abbildung 9.10.:** Erdgeschossplan im Editor mit Treppenbereich

In Abbildung 9.10 ist deshalb jetzt in "Raum 3" ein orange markierter Treppenbereich eingezeichnet, der den kompletten Raum, der die Treppe repräsentiert, einnimmt. Da in diesem Beispiel vom Erdgeschoss aus keine Treppe zu einer tieferliegenden Etage führt, ist der Bereich, in dem eine nach unten führende Treppe sonst liegen würde, als unbetretbarer Bereich (grau markiert) gekennzeichnet. Falls es solch eine nach unten führende Treppe geben würde, müsste natürlich auch dieser (logische) Raum mit einem Treppenbereich gefüllt werden. Auch im Obergeschoss werden natürlich nach analogem Vorgehen Treppenbereiche (bzw. unbetretbare Bereiche) konstruiert, wie in Abbildung 9.11 zu erkennen ist.

Als letzter Schritt muss jetzt noch festgelegt werden, welche Treppenbereiche von jeweils zwei Stockwerken miteinander verbunden sind, damit es den flüchtenden Personen in der Simulation nicht nur möglich ist, einen Treppenbereich zu betreten, sondern über diesen auch auf ein anderes Stockwerk zu gelangen. Wenn man nun wie im hier verwendeten Beispiel das Ergeschoss und das Obergeschoss über eine Treppe miteinander verbinden möchte, klickt man im Erdgeschoss zuerst mit der rechten Maustaste auf die Kante, die das obere En-

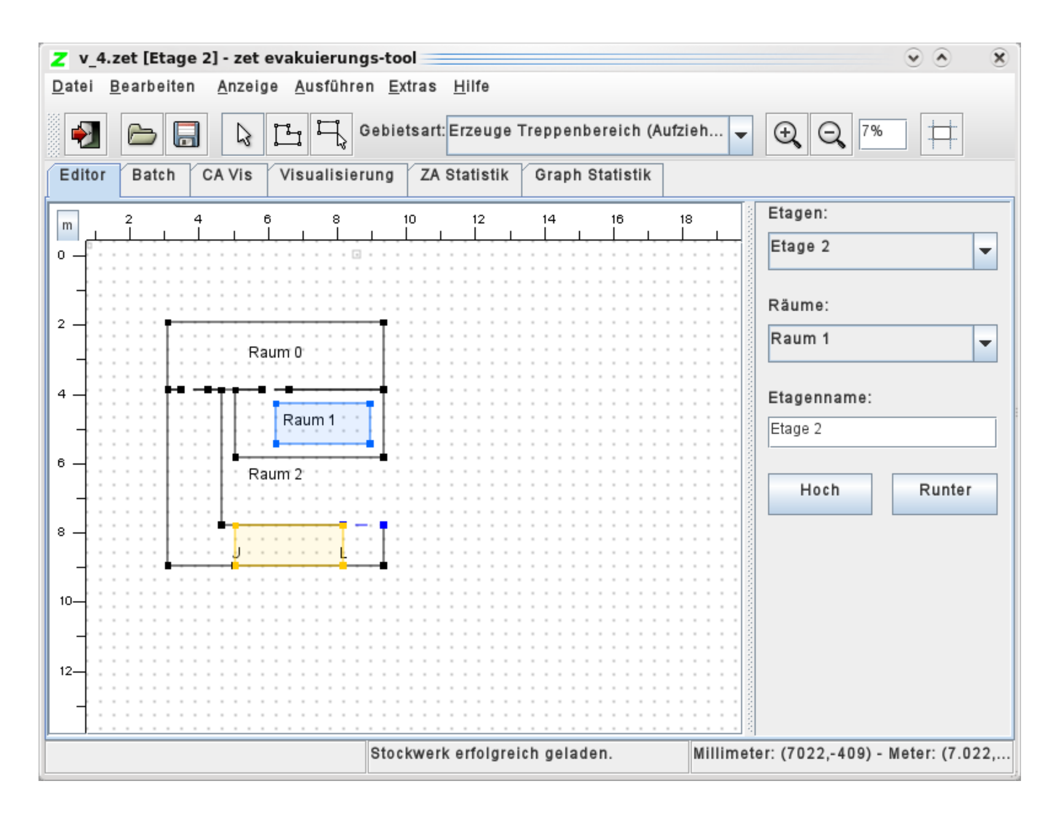



**Abbildung 9.11.:** Obergeschossplan im Editor mit Treppenbereich

de der Treppe repräsentiert, und wählt im dann erscheinenden Menü den Punkt Stockwerkübergang erzeugen. Anschließend wechselt man im Editor auf das Obergeschoss und wählt dort nach einem Rechtsklick auf die untere Kante des dortigen Treppenbereichs ebenfalls im erscheinenden Menü den Punkt Stockwerkübergang erzeugen. Damit sind die beiden Treppenbereiche zu einer logischen Treppe zusammengefügt, die im Modell das Erdgeschoss mit dem Obergeschoss verbindet. Damit ist das Zeichnen eines Plans für das hier verwendete Beispiel abgeschlossen und es kann mit der Simulation begonnen werden.

## 9.4. Belegungen erzeugen und Verwalten

Es existiert ein eigener Editor zum Bearbeiten und Erstellen von Belegungen, der über Bearbeiten | Belegungen... aufgerufen werden kann. Abbildung 9.12 zeigt den Editor. Der Editor ist in drei Bereiche eingeteilt: Der linke Bereich dient zum Verwalten von Belegungen. Eine Belegung entspricht dabei einer Sammlung von verschiedenen Typen, die zu dieser Belegung gehören,

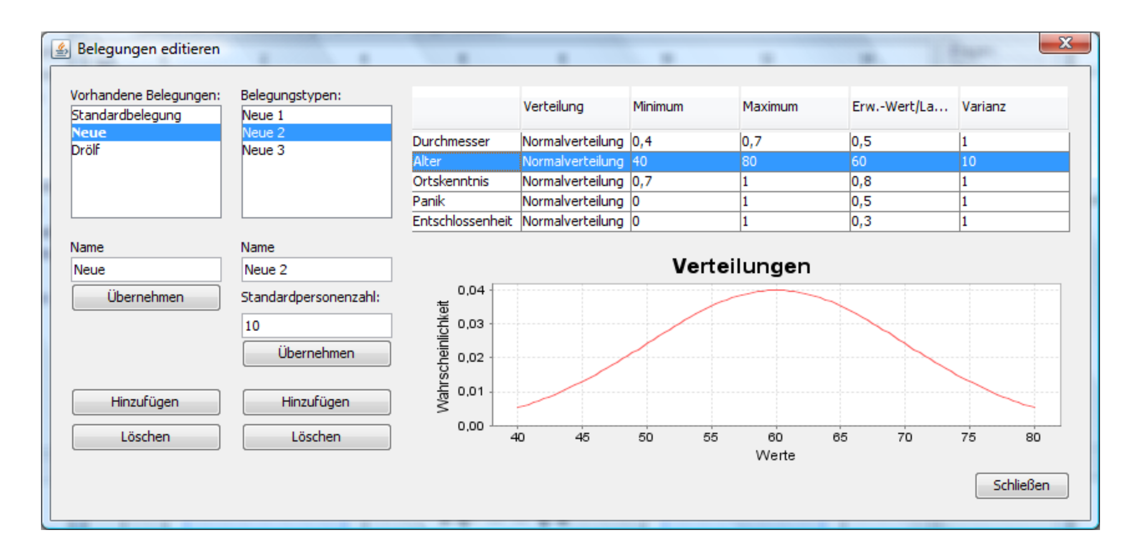



**Abbildung 9.12.:** Der Belegungs-Editor

und einer Menge von Belegungsbereichen, die in der Belegung vorhanden sind. Das Wechseln der Belegung führt dazu, dass die entsprechenden Bereiche im Editor ausgeblendet werden und die zur neuen Belegung gehörenden Bereiche eingeblendet werden. Damit ist es zum Beispiel möglich, verschiedene Personenstrukturen in einer Umgebung zu vergleichen.

Die Schaltflächen Hinzufügen und Übernehmen links unter der Liste der vorhandenen Belegungen erzeugen eine neue Belegung und benennen die aktuell gewählte um. Der benutze Name ist dabei jeweils der im Textfeld Name angegebene. Mit der Schalftfläche Löschen kann eine Belegung komplett gelöscht werden.

Wenn eine Belegung ausgewählt wird, werden die Typen, die in dieser Belegung vorhanden sind, im mittleren Bereich in der Liste angezeigt. Auch hier kann mittels Hinzufügen und Übernehmen ein neuer Typ erzeugt bzw. geändert werden. Zusätzlich zum Namen kann hier im Feld Standardpersonenzahl die gewünschte Zahl der standardmäßig in einem Bereich dieses Typs verteilten Personen gesetzt werden.

Den rechten Bereich bilden eine Tabelle und eine Grafik. Zu jedem Belegungstyp gehören fünf Eigenschaften: **Durchmesser**, **Alter**, **Ortskenntnis**, **Panik** und **Entschlossenheit**. Die Eigenschaften können jeweils durch eine Zufallsverteilung beschrieben werden. In der Spalte Verteilung der Tabelle kann die zugehörige Verteilung ausgewählt werden. Wenn keine Informationen über

die Verteilung bekannt sind, bieten sich eine **Gleichverteilung** oder eine **Normalverteilung** an. In der Tabelle können je nach Verteilung bis zu vier Parameter konfiguriert werden, dabei immer ein **Minimum** und ein **Maximum**. Diese geben die minimal und maximal angenommenen Werte an. So ist es zum Beispiel nicht sinnvoll, ein Alter unter 0 zu haben, was bei normalverteilten Zufallsvariablen durchaus auftreten kann. Auch nach oben kann eine Beschränkung des Alters oder anderer Parameter Sinn machen. **Ortskenntnis**, **Panik** und **Entschlossenheit** stellen dabei prozentuale Anteile dar, das heißt: die Werte müssen zwischen 0 und 1 liegen. Die Größe des Durchmessers wird in Metern angegeben, die Angaben werden jedoch sowohl bei der Optimierung mit Flussalgorithmen auf Graphen als auch bei der Simulation in einem Zellulären Automaten, in dem die Größen festgelegt sind, ignoriert.

Bei der Normalverteilung sind die Parameter 3 und 4 der Erwartungswert und die Varianz, die eine Normalverteilung eindeutig bestimmen. Die gezogenen Zufallszahlen werden um den Erwartungswert herum verteilt, je größer die Varianz, desto breiter wird der genutzte Bereich. Bei der Gleichverteilung gibt es keinen weiteren Parameter. Bei der Exponentialverteilung kann der Parameter $\lambda$, der die Rate beschreibt, gewählt werden; für die Erlang-$k$-Verteilung zusätzlich der Parameter $k$, der die Widerholungen angibt. Für die Hyperexponentialverteilung können die Parameter $\lambda_1$ und $\lambda_2$ gewählt werden, die Gewichtung ist jeweils mit 0,5 fest.

Zur Kontrolle wird die zur Verteilung gehörige Dichtefunktion im Zeichenbereich unten angezeigt.

## 9.5. Weitere Fähigkeiten des Z-Editors

### 9.5.1. Tricks bei der Modellierung

Die Beschränkung auf Räume ohne "Löcher" lässt sich auf einfache Weise umgehen, indem nicht betretbare Bereiche im Inneren von Räumen einfach als unbetretbare Bereiche markiert werden.

Der Fall, in dem Türen im Inneren von Räumen angebracht werden müssen, ist ein besonderer Fall, der dieses Vorgehen sogar notwendig macht, denn offensichtlich können in konvexen Gebieten keine Wände im Inneren sein. Solche

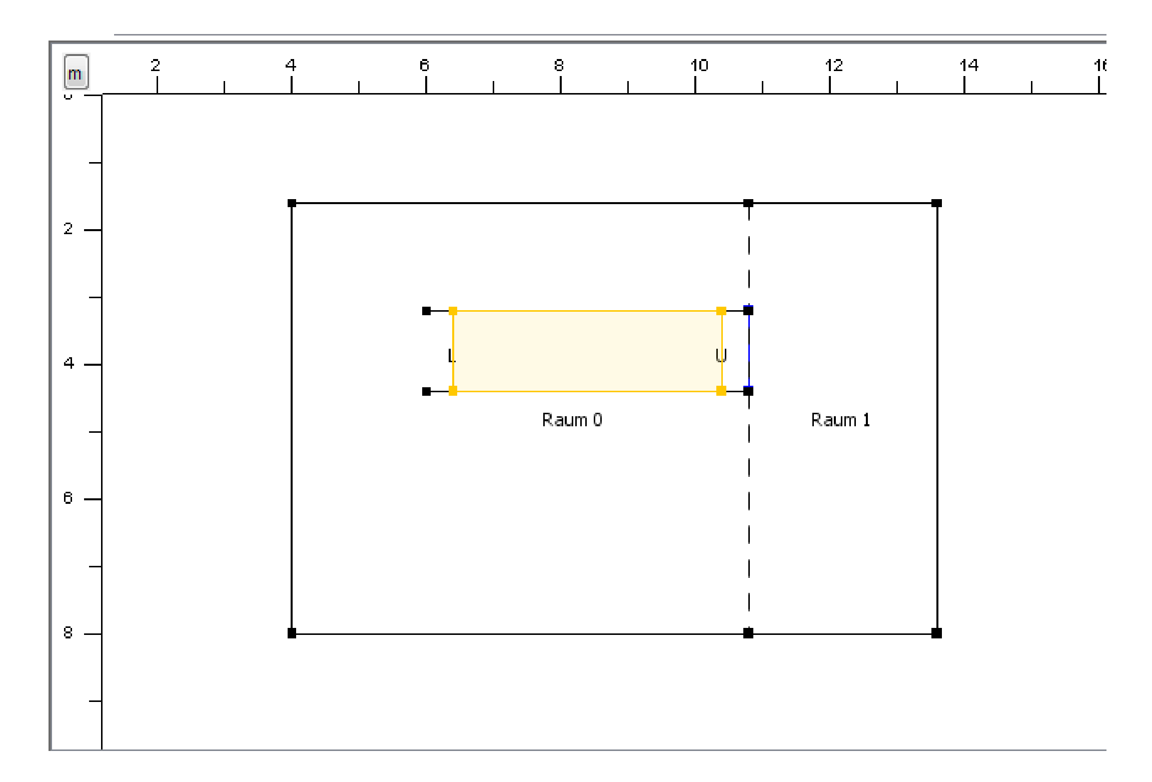



**Abbildung 9.13.:** Beispiel für eine Raumaufteilung. Die Treppe ist etwas kleiner gezeichnet, damit die Barrieren und der dahinter liegende Stockwerkübergang sichtbar sind. Die Räume 0 und 1 simulieren einen großen Raum.

Türen werden z.B. benutzt, um eine freischwebende Treppe zu modellieren. Dazu befindet sich im Inneren des Raumes eine Treppenzone, deren seitliche Ränder von Barrieren eingerahmt werden. Damit wird verhindert, dass Individuen die Treppe von der Seite betreten. Am oberen Ende der Treppe (der noch im Inneren des Raumes liegt) muss, da die Etage gewechselt werden soll, ein Stockwerkübergang erzeugt werden. Indem man den Raum künstlich so aufteilt, dass er am Ende der Treppe endet, kann dort tatsächlich ein Übergang erzeugt werden. Abbildung 9.13 zeigt die Situation. Genauso kann man auch in der Zieletage den Stockwerkübergang nach unten modellieren.

Die Modellierung von Treppen erfordert allgemein spezielle Vorgehensweisen, da Treppenbereiche recht eingeschränkt sind (siehe dazu auch den entsprechenden Abschnitt in 5.1.2). Da sie nur einen Ein- und Ausgang haben, müssen kompliziertere Treppengebilde durch mehrere Einzeltreppen zusammengesetzt werden. Wie bereits erwähnt, können Barrieren genutzt werden, um zu verhindern, dass die Treppe seitlich betreten werden kann. Dies ist zwar bei den meisten Treppen üblich, es gibt aber auch Gegenbeispiele. Treppen in Theatern oder Hörsälen beispielsweise können an jeder Stelle zu den Seiten verlassen und

betreten werden.

Eine besondere Modellierungsmöglichkeit entsteht dadurch, dass man Treppen komplett in eigene Etagen auslagert. Durch die 2-dimensionale Beschränkung der Pläne lassen sich z.B. Leitern oder Wendeltreppen nicht direkt modellieren. Indem nun eine solche „vertikale" Bewegung in unsere Raumrichtung gedreht wird und in eine Zwischenetage verfrachtet wird, kann sie trotzdem modelliert werden. Wendeltreppen werden somit quasi „abgerollt".

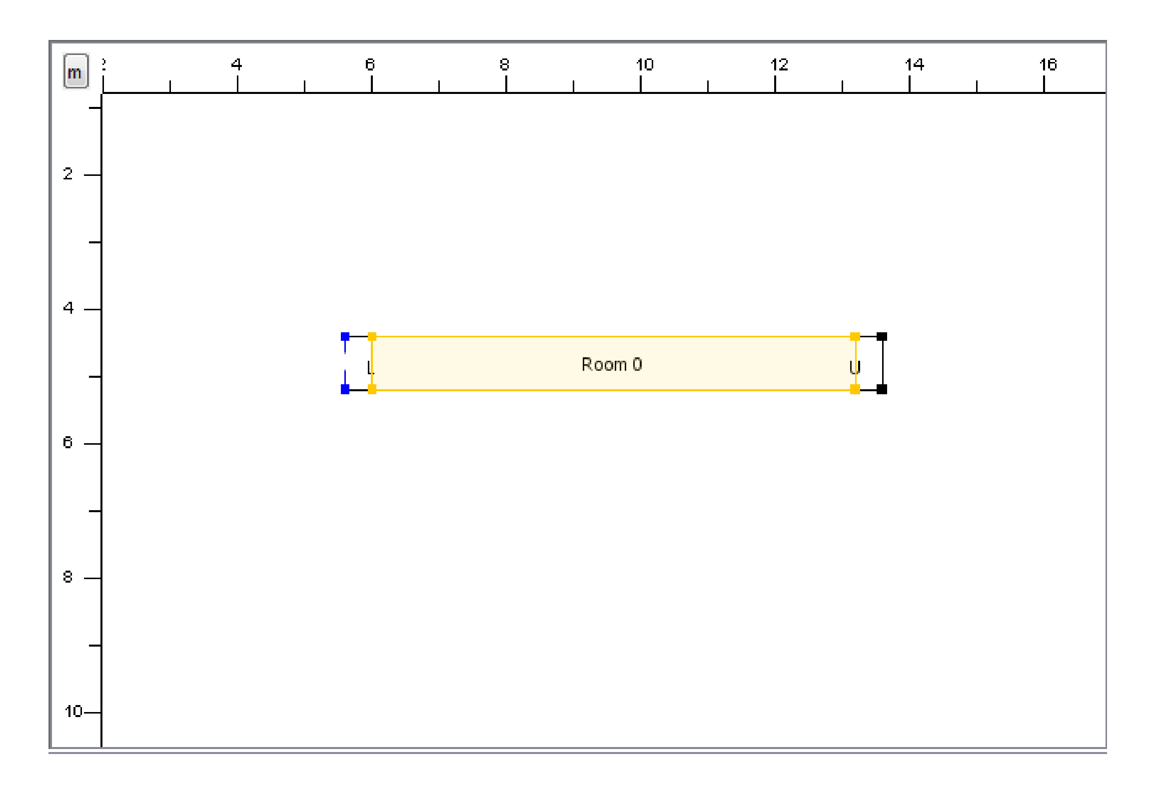



**Abbildung 9.14.:** Die „abgerollte" Wendeltreppe. Personen, welche die Wendeltreppe benutzen, betreten diesen Raum und müssen die ganze Treppe entlanglaufen. Dies entspricht somit der Bewegung nach oben, die in die Ebene abgerollt wurde.

An den Stellen, an der die Treppe beginnt und endet, muss jeweils ein Stockwerkübergang erzeugt werden. Da dies typischerweise in Räumen liegt, muss evtl. das oben beschriebene Vorgehen genutzt werden. Die Übergänge werden aber nicht direkt verbunden, sondern führen jeweils in einen langen, rechteckigen Raum in einer anderen Etage. Dieser Raum repräsentiert die „abgerollte" Treppe und muss komplett mit einem Treppengebiet gefüllt werden. Damit eine Wendeltreppe oder gar eine Leiter realisiert wird, muss natürlich die Geschwindigkeit bzw. der Faktor sehr klein sein.

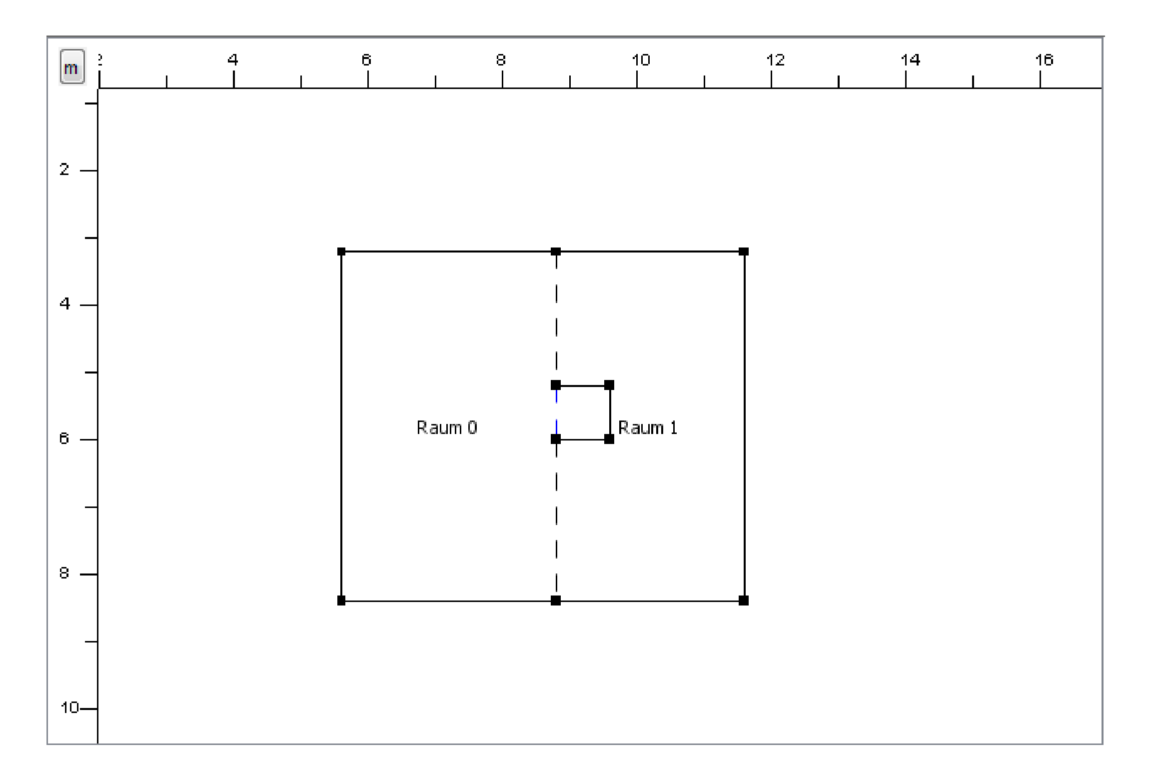



**Abbildung 9.15.:** Zwei verbundene Räume simulieren die Raumaufteilung für eine Wendeltreppe. In der Mitte ist der nicht betretbare Platz, den eine Wendeltreppe einnimmt, ausgespart. Der Stockwerkübergang führt in die spezielle Etage für die Wendeltreppe.

### 9.5.2. Sonstige Features

Es gibt noch einige weitere Features des Z-Editors. Über DATEI || SPEICHERN ALS DXF kann ein Plan im DXF-Format gespeichert werden, welches von vielen CAD-Programmen unterstützt wird. Ein Plan kann mittels DATEI || RASTERN oder über das Symbol RASTERN gerastert werden, d.h. die Eckpunkte werden auf gültige Koordinaten des Rasters verschoben, falls im ungerasterten Modus gemalt worden ist. Weiterhin werden schräge Linien gerastert, d.h. durch einen Polygonzug aus senkrechten und waagrechten Linien ersetzt.

Über die Untermenüpunkte im Menü EXTRAS || BILDPLAN können Pläne als Bild im Hintergrund angezeigt sowie bearbeitet werden. Es können Offsets für die Position(POSITION ANPASSEN...), ein Zoomfaktor (GRÖSSE ANPASSEN...) und die Durchlässigkeit bei der Zeichnung (TRANSPARENZ ÄNDERN...) angepasst werden.

Der Editor kann in mehreren Sprachen angezeigt werden, die dazu benötigten Informationen werden aus Java-Ressource-Dateien gelesen. Standardmäßig

werden die Sprachen **Deutsch** und **Englisch** unterstützt, die aus dem Menü EXTRAS || SPRACHEN ausgewählt werden können.

Weiterhin gibt es einen Editor zum Verwalten von Eigenschaften von Parametern, der Optimierung und der Simulation. Über BEARBEITEN || EIGENSCHAFTEN kann das entsprechende Fenster aufgerufen werden. Im linken Teil befindet sich hier ein Baum, der bestimmte Kategorien und Unterkategorien darstellt, im rechten Teil befinden sich die zur Kategorie gehörenden Werte, die bearbeitet werden können. Mit der Schaltfläche OK werden die Parameter in das laufende Programm übernommen, für den nächsten Start aber nicht gespeichert. Mit der Schaltfläche SPEICHERN können vordefinierte Eigenschaftensätze gespeichert werden, die später wieder geladen werden können. In diesem Fenster kann dazu ein Listenauswahlfeld benutzt werden, falls ein Batch-Durchlauf durchgeführt werden soll, kann ebenfalls für jeden Lauf ein gewünschter Parametersatz ausgewählt werden. Mit SCHLIESSEN kann das Fenster geschlossen werden, ohne dass Änderungen übernommen werden.

# 10. Eine Evakuierung berechnen

Um eine Evakuierung zu starten, müssen Sie zuerst einen neuen Plan zeichnen oder einen bestehenden Plan laden, wie es im Abschnitt 9 erklärt wurde. Dann wechseln Sie im Programm auf die Registerkarte BATCH. Dort können Sie ein Bündel von Evakuierungen zusammenstellen, die Sie alle nach Belieben simulieren oder optimieren lassen können. Der Vorteil bei diesem Verfahren ist, dass Sie sich hinterher in Ruhe die Ergebnisse ansehen können und den PC in der Zwischenzeit einfach rechnen lassen können, ohne immer wieder irgendwelche Eingaben machen zu müssen. Um also Ihren gewählten Gebäudeplan evakuieren zu lassen, drücken Sie auf die Schaltfläche AKTUELLES PROJEKT HINZUFÜGEN. Damit wird das momentan geladene Projekt in die Liste eingefügt.

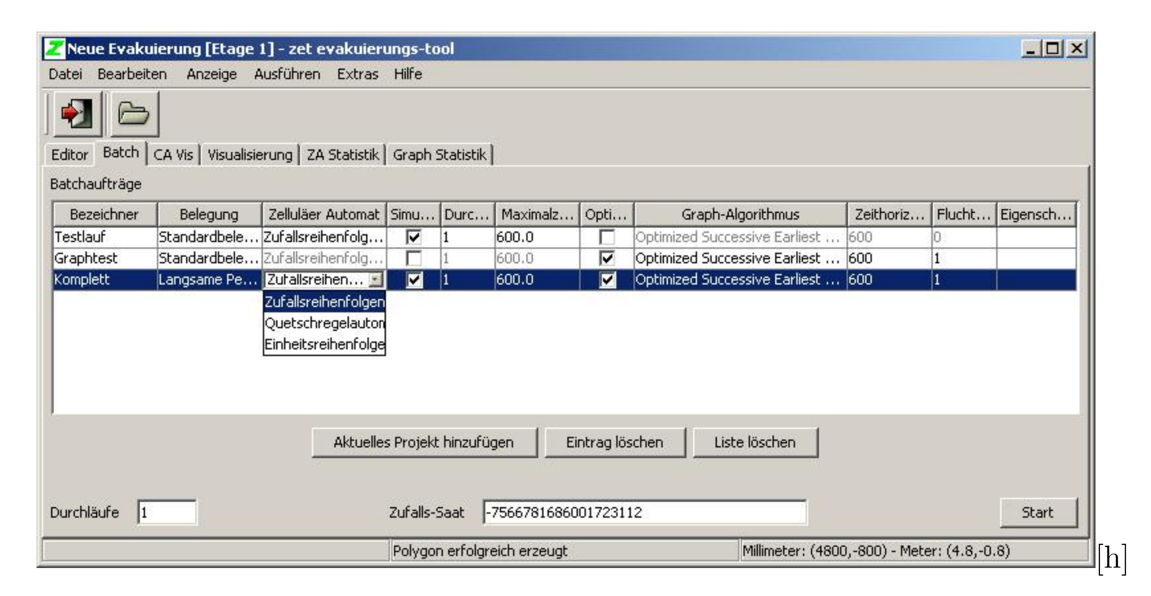
## 10.1. Wahlmöglichkeiten

In der Zeile mit dem eingetragenen Projekt erscheinen nun diverse Auswahlmöglichkeiten, die im Folgenden vorgestellt werden.

**Bezeichner** Dies ist der Name des Eintrags, den Sie hinterher benötigen werden, um die verschiedenen Evakuierungen voneinander zu unterscheiden.

**Belegung** Wenn Sie für das mit dem Eintrag verknüpfte Projekt mehrere Belegungen angegeben haben, können Sie hier auswählen, mit welcher der Belegungen das Gebäude evakuiert werden soll.

**Zellulärer Automat** Hier können Sie auswählen, wie die Simulation der Personen vonstatten gehen soll.



[h]

**Abbildung 10.1.:** Das Batch-Fenster zum Starten der Evakuierungen

**Zufallsreihenfolgenautomat** Die Personen bewegen sich in jedem Zeitschritt in einer zufälligen Reihenfolge. So wird garantiert, dass keine Person bevor- oder benachteilt wird.

**Einheitsreihenfolgenautomat** Die Personen bewegen sich in jedem Zeitschritt in der gleichen Reihenfolge.

**Quetschregelautomat** Die Personen bewegen sich zu jedem Zeitschritt in einer zufälligen Reihenfolge. Außerdem können sie sich aneinander vorbei "quetschen" falls 2 Personen direkt voreinander stehen und die Plätze tauschen wollen.

**Simulation** Ob die Evakuierung mit einem zellulären Automaten simuliert werden soll.

**Durchläufe** Wie oft die Evakuierung simuliert werden soll (mehr Durchläufe bringen sicherere Ergebnisse in der Statistik).

**Maximalzeit** Personen, die das Gebäude bis zu dieser Zeit (in Sekunden) nicht verlassen haben, werden als "nicht evakuiert" gewertet.

**Optimierung** Ob die optimale Evakuierung als Graphfluss berechnet werden soll.

**Graph-Algorithmus** Der bei der Optimierung zu benutzende Graph-Flussalgorithmus. Die einzelnen Algorithmen weisen sehr stark unterschiedliche

Laufzeiten und Anwendungsmöglichkeiten auf. Für die Simulation der Evakuierung sollten Sie Transshipments nehmen (hier standardmäßig das EARLIEST ARRIVAL TRANSSHIPMENT), für eine Betrachtung des maximalen Durchflusses durch ein Gebäude können Sie z.B. MAXFLOW nehmen.

**Zeithorizont** Die obere Grenze für die betrachtete Zeit bei der Optimierung. Hat bei Transshipments dieselbe Funktion wie die Maximalzeit für die Simulation, bei MaxFlow ist der Zeithorizont die Zeit während der die Personen das Gebäude durchlaufen können, allerdings ohne Anlauf- oder Ablaufeffekte. Die Personen sind bei MaxFlow ein konstanter Strom, es soll ja auch nur der Durchfluß gemessen werden und nicht die Zeit für eine konkrete Evakuierungssimulation.

**Fluchtplan-Simulationen** Ob der berechnete Fluchtplan im zellulären Automaten simuliert werden soll und wenn ja, wie oft. Dies kann benutzt werden um die Ergebnisse der Graphalgorithmen zu validieren und stellt dann eine untere Schranke für die Evakuierungszeit dar, da sich die Personen in den dann stattfindenden ZA-Durchläufen immer auf den idealen Wegen zum Ausgang begeben.

**Eigenschaftsdatei** Hier können spezielle Eigenschaften der ausgewählten Durchläufe festgelegt werden, z.B. eine Simulation ohne Berücksichtigung der Panik.

Tipp: Achten Sie immer darauf, dass Sie Ihre Eingaben (z.B. die Anzahl der Durchläufe) mit der RETURN-Taste bestätigen bevor Sie auf START klicken.

## 10.2. Mehrere Projekte in einem Batch

Wenn Sie mehrere Projekte in einem Batch zusammen berechnen lassen wollen (was ja der Sinn und Zweck der Batch-Einrichtung ist), dann müssen Sie Folgendes beachten:

- Änderungen, die Sie am geladenen Projekt durchführen, wirken sich auch auf die bereits zum Batch hinzugefügten Einträge aus. D.h. wenn Sie z.B. 2 Durchläufe machen wollen - einen ohne einen Extra-Ausgang und einen

zweiten mit diesem Ausgang - dann sollten Sie diese beiden Konfigurationen des Gebäudes in 2 verschiedenen Dateien abspeichern bevor Sie mit der Erstellung des Batch beginnen. Wenn Sie nämlich das Gebäude erst einmal ohne den Extra-Ausgang zeichnen, dann dem Batch hinzufügen, dann den Ausgang einzeichnen und das Ergebnis wieder dem Batch hinzufügen, dann werden hinterher beide Einträge diesen Extra-Ausgang haben, weil sie beide mit demselben geladenen Projekt verknüpft sind.

- Wenn Sie zu einem Gebäude mehrere Belegungen gespeichert haben und alle Simulationen gleichzeitig starten wollen, brauchen Sie obiges Vorgehen jedoch nicht. Dann können Sie einfach immer wieder das gleiche geladene Projekt zum Batch hinzufügen und im Batch-Fenster die gewünschten Belegungen wählen.

## 10.3. Weitere Bedienelemente

Unten im Fenster finden Sie dann noch eine Box, in der Sie die Anzahl Durchläufe für die Simulation für alle bereits angelegten Einträge festsetzen können. Weiterhin baut die Simulation der Evakuierung natürlich großteils auf Zufallsdaten auf; wenn Sie also einen bestimmten Ablauf der Evakuierung im Nachhinein noch einmal genauso wiederholen möchten, so können Sie unten in der Box mit der Beschriftung ZUFALLS-SAAT den Saatwert des verwendeten Zufallsgenerators einstellen. Diese Möglichkeit ist für den normalen Benutzer allerdings ohne große Bedeutung.

## 10.4. Start

Wenn Sie Ihren Batch fertig angelegt haben, dann können Sie per Druck auf die Schaltfläche START unten rechts alle gewählten Berechnungen durchführen lassen. Es werden diverse Fortschrittsdialoge zu sehen sein, ganz am Ende müssen Sie den letzen dieser Dialoge mit SCHLIESSEN schließen und können dann die Ergebnisse der Berechnungen in den Registerkarten VISUALISIERUNG, ZA STATISTIK und GRAPH STATISTIK betrachten.

## 10.5. Speichern und Laden von Batch-Ergebnissen

Sie können die Ergebnisse Ihrer im Batch gewählten Berechnungen speichern, indem Sie nach Fertigstellung der Ergebnisse im Datei-Menü des Programms auf Speichere Ergebnisse... gehen. Beachten Sie hierbei bitte, dass Sie die *Ergebnisse speichern* müssen *bevor Sie die Visualisierung betrachten*, ansonsten wird die Visualisierung nach dem Laden der Ergebnisdatei nicht funktionieren. Wenn Sie andersherum eine Ergebnisdatei laden wollen, können Sie dies dort auch tun, unter Öffne Ergebnisdatei. . . . Die geladenen Ergebnisse stehen dann auf den Registerkarten Visualisierung, ZA Statistik und Graph Statistik zur Verfügung.

# 11. Eine Evakuierung auswerten

Wenn Sie (wie in Kapitel 10 beschrieben) eine oder mehrere Evakuierungen durchgeführt haben, können Sie die Ergebnisse in den Registerkarten ZA Statistik und Graph Statistik betrachten.

## 11.1. ZA-Statistik

Die ZA-Statistik umfasst verschiedenste Statistiken über Individuen und ihr Verhalten. Diese werden in Form von Balken-, Linen- sowie Kuchendiagrammen dargestellt. Nachfolgend werden die verschiedenen Typen erläutert und beschrieben, wie sie zu nutzen sind.

### 11.1.1. Diagrammtypen

Bei allen Diagrammen wird grundsätzlich immer der Durchschnitt über alle Durchläufe berechnet.

#### Statische Diagrammtpyen

Diese Art von Diagrammen geben ausschließlich Werte bzw. Informationen über einen Bezeichner wieder. Will man die Informationen zweier Bezeichner miteinander vergleichen, muss man einfach für beide Bezeichner je ein Diagramm von diesem Typ erzeugen. ZET enthält nur ein statisches Diagramm:

**Grundinformationen** Tabelle von Informationen zu einem Bezeichner. Die Informationen, die man erhält, geben folgende Werte an:

**Informationen für Modell** gibt den Bezeichner an, über den diese Tabelle informiert.

**Evakuierungszeit in Sekunden** gibt in Sekunden an, wielange es gedauert hat, bis die Simulation beendet wurde (siehe dazu *Ende der Simulation* in Abschnitt 7.1.5).

**Evakuierungszeit in ZA-Schritten** gibt dieselbe Zeit in Zeitschritten des Zellulären Automaten an.

**Anzahl Individuen** ist die Gesamtanzahl der zu evakuierenden Individuen.

**evakuiert** ist die Anzahl der Individuen, die sich in Sicherheit bringen konnten (siehe dazu auch *Gerettete Individuen* in Abschnitt 7.1.5).

**nicht evakuiert** gibt die Gesamtanzahl der Individuen an, die nicht gerettet wurden (siehe dazu auch *Nicht gerettete Individuen* in Abschnitt 7.1.5).

**nicht evakuiert weil kein Ausgang erreichbar** gibt die Anzahl der Individuen an, die nicht gerettet werden konnten, weil sie eingeschlossen waren.

**nicht evakuiert weil die Zeit nicht gereicht hat** gibt die Anzahl der Individuen an, die nicht gerettet werden konnten, weil sie es in der eingegebenen Maximalzeit nicht bis zu einem sicheren Bereich oder Ausgang geschafft haben.

**beste Evakuierungszeit (Durchlaufindex,Zeit)** gibt zuerst die Nummer des Durchlaufs an, in dem die beste Evakuierungszeit erreicht wurde und dann diese Evakuierungszeit in Sekunden.

**durchschnit. Evakuierungszeit (Durchlaufindex,Zeit)** gibt zuerst die Nummer des Durchlaufs an, dessen Evakuierungszeit am nächsten an der durchschnittlichen Evakuierungszeit aller Durchläufe liegt und dann die Evakuierungszeit dieses Durchlaufs in Sekunden.

**schlechteste Evakuierungszeit (Durchlaufindex,Zeit)** gibt zuerst die Nummer des Durchlaufs an, in dem die schlechteste Evakuierungszeit erreicht wurde und dann diese Evakuierungszeit in Sekunden.

### Diagrammtypen für eine Datenreihe

Mit diesen Diagrammtypen kann man Informationen für eine Datenreihe, aber nicht für mehrere Datenreihen gleichzeitig, anzeigen. Möchte man die Informationen für mehrere Datenreihen vergleichen, muss man mehrere Diagramme des Typs erzeugen.

**Ausgangsverteilung** Dies ist ein Kuchendiagramm, das die prozentuale Verteilung der Individuen auf die verschiedenen Ausgänge widerspiegelt.
Ausgänge können im Editor mit Namen versehen werden, die hier entsprechend angezeigt werden. Die im Editor automatisch generierten Evakuierungsausgänge werden hierbei mit *Standardausgang + Nr.* benannt.
Geht man mit der Maus auf einen Teil des Kuchendiagrammes, wird die absolute Anzahl an Individuen angezeigt, die diesen Ausgang genommen haben. Bei mehreren Durchläufen ist dies die Summe der Individuen aus allen Durchläufen.
Individuen, die nicht gerettet werden konnten, gehen in dieses Diagramm nicht mit ein.

**evakuierte Individuen in Prozent** In diesem Kuchendiagramm kann man sehen, wieviel Prozent der Individuen der ausgewählten Datenreihe evakuiert und wieviel Prozent nicht gerettet wurden.
Geht man mit der Maus auf einen Teil des Kuchendiagrammes, wird die genaue Prozentzahl an Individuen angezeigt, die evakuiert bzw. nicht evakuiert wurden.

### Diagrammtypen für eine oder mehrere Datenreihen

Diese Diagramme lassen eine Auswertung verschiedener Statistiken für eine oder auch mehrere Datenreihen im Vergleich zu. Je nach Statistik handelt es sich entweder um einen Wert über die gesamte Simulationsdauer, der als Balkendiagramm dargestellt wird, oder um eine Abbildung des Wertes über die verstrichene Zeit. Für jede Datenreihe wird ein Balken bzw. ein Funktionsgraph erzeugt. Bei beiden Diagrammarten kann man sich genaue Wert anzeigen lassen, wenn man mit der Maus über den Balken bzw. über eine Stelle des Funktionsgraphen fährt. Man kann in den Diagrammen auch zoomen:

Dazu wird entweder mit der Maus ein Rechteck gezogen oder mit der rechten Maustaste in das Diagramm geklickt. Es erscheint dann ein Kontektmenü, in dem Heraus- oder Hereinzoomen gewählt werden kann.

**minimale / durchschnittliche / maximale Blockadezeit** stellt das Minimum, den Durchschnitt bzw. das Maximum über die Blockadezeiten der Individuen der Datenreihe als Balkendiagram dar. Ein Individuum gilt als blockiert, wenn es in einem Zeitschritt stehenbleibt, weil es keine freien Zellen um es herum gibt bzw. weil es nicht in die gewünschte Richtung gehen kann („nach vorne“ hin keine freien Zellen). Wenn ein Individuum trödelt und deshalb stehenbleibt, gilt diese Zeit nicht als Blockadezeit.
Die Zeit, die ein Individuum stehenbleibt, wenn es in einem Zeitschritt blockiert wird, hängt von der gewählten Maximalgeschwindigkeit ab. Ein Zeitschritt in der Simulation dauert 0,4/Maximalgeschwindigkeit Sekunden.
Individuen, die nicht gerettet werden konnten, weil kein Ausgang erreichbar war, gehen in die Berechnung nicht mit ein.

**zurückgelegte Distanz** stellt den Durchschnitt über die zurückgelegte Strecke der Individuen der Datenreihe als Balkendiagram dar.
Individuen, die nicht gerettet werden konnten, weil kein Ausgang erreichbar war, gehen in die Berechnung nicht mit ein.

**minimale Distanz zum initialen / nächsten Ausgang** stellt den Durchschnitt über die minimalen Distanzen der Anfangspositionen der Individuen der Datenreihe zum initialen /nächsten Ausgang als Balkendiagram dar.
Die minimale Distanz ist dabei die kürzeste Strecke, die das Individuum laufen kann, um zu dem Augang zu gelangen. Der initiale Ausgang ist der Ausgang, dessen Potential das Individuum zu Beginn der Simulation zugewiesen bekommt. Der nächste Ausgang ist der Ausgang, der am nächsten an der Anfangsposition des Individuums liegt.
Individuen, die nicht gerettet werden konnten, weil kein Ausgang erreichbar war, gehen in die Berechnung nicht mit ein.

**minimale/durchschnittliche/maximale Zeit bis Sicher** stellt das Minimum bzw. den Durchschnitt bzw. das Maximum über die Zeit, die die Individuen der Datenreihe benötigt haben, um gerettet zu werden, als Balken-

diagram dar.
Individuen, die nicht gerettet werden konnten, gehen in die Berechnung nicht mit ein. Werden in einem Durchlauf einige Individuen der Datenreihe gerettet, in einem anderen Durchlauf jedoch wird kein Individuum gerettet, geht letzterer nicht in die Berechnung des Durchschnitts über die Durchläufe mit ein. Werden alle Individuen einer Datenreihe in allen Durchläufen nicht gerettet, erhält diese Gruppe den Wert 0.

**durchschnittliche/maximale Geschwindigkeit** stellt jeweils den Durchschnitt über die durchschnittliche bzw. maximale Geschwindigkeit der Individuen der Datenreihe als Balkendiagram dar.
Individuen, die nicht gerettet werden konnten, weil kein Ausgang erreichbar war, und Individuen, die sich schon zu Beginn der Simulation in einem Bereich aufhielten, in dem sie gerettet waren, gehen in die Berechnung nicht mit ein.

**Ankunftskurve** trägt die Anzahl an evakuierten Individuen der Datenreihe über die Zeit ab.

**Distanz über Zeit** trägt den Durchschnitt über die bisher zurückgelegte Distanz der Individuen der Datenreihe über die Zeit ab.
Individuen, die nicht gerettet werden konnten, weil kein Ausgang erreichbar war, gehen in die Berechnung nicht mit ein.

**durchschnittliche/maximale Geschwindigkeit über Zeit** trägt den Durchschnitt bzw. das Maximum über die aktuellen Geschwindigkeiten der Individuen der Datenreihe über die Zeit ab.
Individuen, die nicht gerettet werden konnten, weil kein Ausgang erreichbar war, gehen in die Berechnung nicht mit ein. Individuen, die zu einem bestimmten Zeipunkt schon gerettet sind, gehen in die Berechnung für diesen Zeitpunkt nicht mit ein. Sind in einem Durchlauf zu einem bestimmten Zeitpunkt noch nicht alle Individuen der Datenreihe gerettet, in einem anderen Durchlauf jedoch schon, geht letzterer nicht in die Berechnung des Durchschnitts über die Durchläufe mit ein. Sind alle Individuen einer Datenreihe in allen Durchläufen zu einem bestimmten Zeitpunkt bereits gerettet, bricht der Funktionsgraph an dieser Stelle ab.

**Panik über Zeit** trägt den Durchschnitt über die Panikwerte der Individuen der Datenreihe über die Zeit ab.

Die Bestimmung, welche Werte in die Berechnung mit eingehen, erfolgt wie in „durchschnittliche/maximale Geschwindigkeit über Zeit“.

**Erschöpfung über Zeit** trägt den Durchschnitt über die Erschöpfungswerte der Individuen der Datenreihe über die Zeit ab.
Die Bestimmung, welche Werte in die Berechnung mit eingehen, erfolgt wie in „durchschnittliche/maximale Geschwindigkeit über Zeit“.

### 11.1.2. Diagramme erzeugen

In der rechten Seite des Fenster kann man Einstellungen vornehmen, um sich Diagramme anzeigen zu lassen. Diese erscheinen in der linken Seite des Fensters nebeneinander.

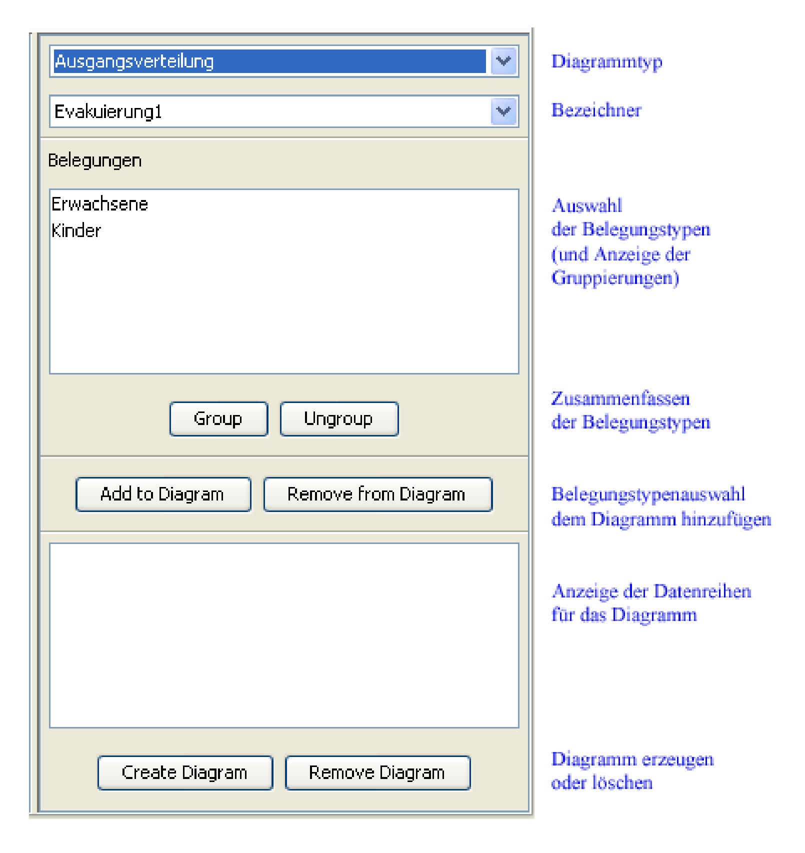



**Abbildung 11.1.:** Übersicht Bedienelemente ZA-Statistik

Die Bedienelemente zum Erzeugen von Diagrammen befinden sich in der rechten Seite des Fensters (siehe Abbildung 11.1).

Um ein **statisches Diagramm** zu erzeugen, sind folgende Schritte notwendig:

1. *Diagrammtyp auswählen*: Dies geschieht durch die oberste Drop-Down-Box.
2. *Bezeichner auswählen*: Die zuvor im Batchfenster festgelegten Namen für jede ZA-Simualtion tauchen hier wieder auf.
3. *Diagramm erzeugen*: Nach einem Klick auf CREATE DIAGRAM erscheint das Diagramm in der linken Seite des Fensters.

Um ein Diagramm vom Typ **„Diagramm für eine Datenreihe“** zu erzeugen, sind folgende Schritte notwendig:

1. *Diagrammtyp auswählen*: siehe oben
2. *Bezeichner auswählen*: siehe oben
3. *Belegungstypen auswählen*: Nachdem man einen Bezeichner ausgewählt hat, erscheinen in der Liste „Belegungen“ die zu diesem ZA (und der im Batchfenster gewählten Belegung) gehörigen Belegungstypen. In dieser Liste können Sie nun genau einen Eintrag auswählen. Mehrere Einträge lassen sich durch den GROUP-Button zu einem verschmelzen und werden durch den UNGROUP-Button wieder getrennt (siehe auch 11.1.3).
4. *Datenreihe für Diagramm verwenden*: Mit einem Klick auf ADD TO DIAGRAM erzeugen sie das Diagramm. Im Titel des Diagrammes erscheint der Name der ausgewählten Datenreihe in der Form „Bezeichner - Belegungstyp(en)name(n)“.

Um ein Diagramm vom Typ **„Diagramm für eine oder mehrere Datenreihen“** zu erzeugen, sind folgende Schritte notwendig:

1. *Diagrammtyp auswählen*: siehe oben
2. *Bezeichner auswählen*: siehe oben
3. *Belegungstypen auswählen*: In der Liste „Belegungen“ können Sie nun einen oder mehrere Einträge (Shift- oder Strg-Taste gedrückt halten) auswählen. Mehrere Einträge lassen sich durch den GROUP-Button zu einem verschmelzen und werden durch den UNGROUP-Button wieder getrennt (siehe auch 11.1.3).

4. *Datenreihe(n) dem Diagramm hinzufügen*: Sobald Sie eine Auswahl getroffen haben, lässt sie sich dem Diagramm durch einen Klick auf ADD TO DIAGRAM hinzufügen. Die hinzugefügten Einträge erscheinen in der Datenreihen-Liste je in der Form „Bezeichner - Belegungstyp(en)name(n)“.

5. *Weitere Datenreihen hinzufügen*: Die Schritte 2, 3 und 4 lassen sich nun wiederholen, um weitere Datenreihen (evtl. auch aus anderen Bezeichnern) zu generieren.

6. *Diagramm erzeugen*: Nach einem Klick auf CREATE DIAGRAM erscheint das zusammengestellte Diagramm in der linken Seite des Fensters.

### Leere oder nicht erzeugte Diagramme

Wenn ein leeres Diagramm erscheint oder kein Diagramm erscheint, kann das mehrere Ursachen haben:

- Eventuell wurde beim Erstellen eine falsche Einstellung gewählt. Erstellen Sie das Diagramm erneut und beachten Sie die oben angegebene Reihenfolge.
  Es ist auch zu Beachten, dass die Datenreihe, die Sie mit ADD TO DIAGRAM hinzufügen möchten, vorher in der Liste „Belegungen“ *markiert* sein muss.
  Bei den Diagrammen für nur eine Datenreihe kommt es zu Fehlern, wenn Sie mehr als eine Datenreihe markiert haben.
  Auch gleiche Namen der Datenreihen (durch gleiche Namen der Bezeichner und der Belegungstypen) können zu Fehlern führen, so dass sie z.B. ein Diagramm nur für eine dieser Datenreihen bekommen.

- Eventuell haben Sie ein Diagramm für eine Gruppe von Individuen erstellt, die gar keine Individuen enthält. Das kann vorkommen, wenn einem Belegungstyp im Editor keine Belegungsbereiche zugeordnet wurden oder alle Belegungsbereiche dieses Belegungstyps 0 Individuen enthalten. Im Titel des leeren Diagrammes erscheint in diesem Fall die Überschrift *NO INDIVIDUALS in at least one of the chosen dataset(s).*
  Wenn Sie mehrere Datenreihen vergleichen, von denen nicht alle Datenreihen Individuen enthalten, kann es zu falscher Beschriftung der Werte

kommen. Achten Sie also darauf, keine Belegungstypen mit 0 Individuen anzulegen.

- Eventuell ist das Diagramm überhaupt nicht leer, sondern stellt nur den Wert 0 (Balkendiagramm) bzw. die Nullfunktion (Funktionsgraph) dar. Dies kann vor allem bei Diagrammen passieren, die einen minimalen Wert berechnen wie zum Beispiel die minimale Blockadezeit. Dass dieser Wert 0 ist, bedeutet nur, dass es (in jedem Durchlauf) mindestens ein Individuum gegeben hat, das nie blockiert wurde.
  Auch wenn man versucht, für eine Gruppe von eingeschlossenen Individuen einen Wert wie Distanz zu einem Ausgang zu berechnen, erhält man den Wert 0.

### 11.1.3. Datenreihen erstellen und vergleichen

**Datenreihen erstellen:** Eine Datenreihe ist eine Reihe von Daten für eine bestimmte Gruppe von Individuen (welche Daten das sind, bestimmt natürlich das ausgewählte Diagramm). Diese Gruppe von Individuen kann minimal aus allen Individuen eines Belegungstyps (eines Bezeichners) bestehen.
Man kann aber auch die Individuen mehrerer Belegungstypen desselben Bezeichners zusammenfassen und so ihre Daten zu einer einzigen Datenreihe vereinigen. Dies geschieht über den Group-Button. Wählen Sie dazu mehrere Zeilen in der Liste „Belegungen“ aus (Shift- oder Strg-Taste gedrückt halten) und klicken Sie auf den Group-Button. Die Zeilen vereinigen sich zu einer Zeile, d.h. Datenreihe. Mit dem Ungroup-Button lassen sich die Datenreihen wieder aufsplitten.
Tipp: Achten Sie darauf, dass Sie nach einer Group- oder Ungroup-Aktion erst wieder auf die Datenreihe klicken müssen (da sie nicht mehr ausgewählt ist) um sie der Datenreihenliste hinzufügen zu können.
Eine Zeile in der Liste „Belegungen“ steht also für eine Datenreihe, die evtl. nur aus den Individuen eines Belegungstypen oder auch aus den Individuen mehrerer Belegungstypen generiert wird.

**Datenreihen vergleichen:** In allen Diagrammen des Typs „Diagramme für eine oder mehrere Datenreihen“ können beliebig viele Datenreihen miteinander

verglichen werden. Es können sogar Datenreihen, die zu unterschiedlichen Bezeichnern gehören, verglichen werden. Um Datenreihen zu vergleichen, erstellt man erst eine Liste der Datenreihen, die man vergleichen möchte. Sie wird im unteren Teil des Bedienfensters angezeigt. Wenn man mehrere Datenreihen desselben Bezeichners zu der Liste hinzufügen möchte, kann man diese Datenreihen in der oberen Liste auch gemeinsam markieren (Shift- oder Strg-Taste gedrückt halten) und mit einem einzigen Klick auf Add to Diagram in die Datenreihenliste übernehmen (anstatt jede Datenreihe einzeln übernehmen zu müssen). Wenn man Datenreihen von unterschiedlichen Bezeichnern vergleichen möchte, muss man erst die Datenreihen des einen Bezeichners auswählen und mit Add to Diagram übernehmen und dann die Datenreihen des anderen anderen Bezeichners.
Es ist ebenfalls möglich, eine Datenreihe beliebig oft in die Datenreihenliste einzutragen.
Mit dem Remove from Diagramm- Button kann man ausgewählte Datenreihen auch wieder aus der Liste entfernen.

## 11.2. Graphstatistik

Die Graph-Statistik bietet die Möglichkeit durch graphbasierte Algorithmen berechnete Ergebnisse in eine kompakte, von Benutzer spezifizierbare Form zu bringen. Der Benutzer kann dabei wählen, welche statistischen Größen dargestellt werden sollen, wie sie zusammengefasst werden sollen und auf welche Weise sie dargestellt werden.

### 11.2.1. Benutzerinterface

Das Benutzerinterface der Graphstatistik besteht aus zwei großen Bereichen. Zum einen dem Anzeigebereich auf der linken Seite, wo die gewünschten Diagramme und Tabellen dargestellt werden, zum anderen dem Einstellungsbereich auf der rechten Seite, der sich nochmals in Unterbereiche für Profile, Diagramme und statistische Größen aufteilt. Abbildung 11.2 zeigt die Bereiche der Graphstatistik.

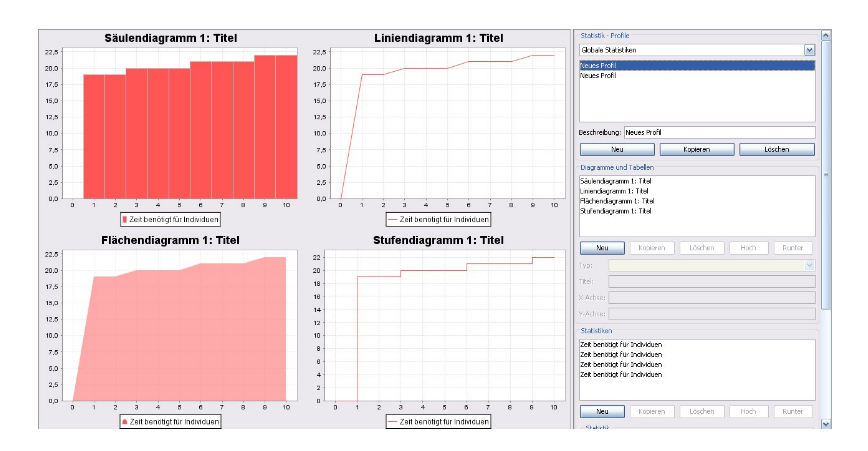



**Abbildung 11.2.:** Die Graphstatistik

## 11.2.2. Statistische Größen

Die statistischen Größen sind in 4 Kategorien aufgeteilt, basierend auf der Modellkomponente, zu der sie gehören. Es gibt daher globale Größen, sowie auf Knoten-, Kanten- und Fluss bezogene Größen.

### Globale Größen

Es werden zwei globale Größen unterstützt:

**Insgesamt benötigte Zeit** gibt die Zeit an, die für die Evakuierung aller Individuen nötig war.

**Zeit benötigt für Individuen** gibt für jede Individuen-Anzahl an, wieviel Zeit benötigt wurde, um sie zu evakuieren.

### Knotengrößen

Für jeden Knoten (und jede Menge von Knoten, siehe dazu den nächsten Abschnitt) werden folgende Statistiken unterstützt:

**Gemittelte Auslastung** gibt das Verhältnis von im Knoten gespeicherten Fluss zu seiner Kapazität an, über die Zeit gemittelt.

**Eingehende Flussrate** gibt die Rate an, mit der Fluss in den Knoten zu einem Zeitpunkt eingeht.

**Eingehende Flussmenge** gibt die Menge an Fluss an, die bis zu einem Zeitpunkt in den Knoten eingegangen ist.

**Ausgehende Flussrate** gibt die Rate an, mit der Fluss den Knoten zu einem Zeitpunkt verlässt.

**Ausgehende Flussmenge** gibt die Menge an Fluss an, die bis zu einem Zeitpunkt den Knoten verlassen hat.

**Speicherrate** gibt die Rate an, mit der sich der im Knoten gespeicherte Fluss zu einem Zeitpunkt verändert.

**Gespeicherter Fluss** gibt die Menge an Fluss an, die zu einem Zeitpunkt in dem Knoten gespeichert ist.

**Gespeicherte Flussmenge** gibt die Menge an Fluss an, die bis zu einem Zeitpunkt in dem Knoten gespeichert wurde.

**Auslastung** gibt das Verhältnis von im Knoten gespeicherten Fluss zu seiner Kapazität an.

#### Kantengrößen

Für Kanten und Kantenmengen werden folgende Größen zur Auswertung angeboten:

**Gemittelte Auslastung** gibt das Verhältnis von dem Fluss, der in die Kante eingeht zu der Kapazität der Kante an, über die Zeit gemittelt.

**Kapazität** gibt die Kapazität der Kante an.

**Flussrate** gibt die Rate an, mit der Fluss in die Kante zu einem Zeitpunkt einfließt.

**Flussmenge** gibt die Menge an Fluss an, die bis zu einem Zeitpunkt in die Kante geflossen ist.

**Auslastung** gibt das Verhältnis von dem Fluss, der in die Kante eingeht zu der Kapazität der Kante an.

#### Flussgrößen

Für Flusseinheiten und Mengen von Flusseinheiten werden folgende Statistiken unterstützt:

**Gesamtfahrzeit** gibt die Zeit an, in der sich die Flusseinheit bewegt hat.

**Gewartete Zeit** gibt die Zeit an, in der die Flusseinheit gewartet hat.

**Gesamtzeit** gibt die Zeit an, die die Flusseinheit bis zum Erreichen der Senke gebraucht hat.

**Entfernung zur Senke** gibt die kürzeste Entfernung zur von der Flusseinheit benutzten Senke an.

**Entfernung zur nächsten Senke** gibt die kürzeste Entfernung zur der Senke an, die der Flusseinheit am nächsten ist.

### 11.2.3. Zusammenfassen von Statistik-Größen

Die oben aufgeführten Größen lassen sich nicht nur für einzelne Knoten, Kanten und Flusseinheiten berechnen, sondern auch für Mengen dieser Objekte. Für diesen Fall bietet die Statistik mehrere Optionen an, wie aus den Statistiken für die einzelnen Objekte die Statistiken der Menge berechnet werden.

Ebenso ist es möglich, dass mehrere Durchläufe eines Algorithmus berechnet wurden, und nun für jeden Knoten / Kante / etc. multiple Werte vorliegen – auch diese können auf benutzerspezifizierte Weise zusammengefasst werden.

Es werden dabei folgende Arten der Zusammenfassung unterstützt:

**Arithmetrisches Mittel** bildet das arithmetrische Mittel aus allen Werten.

**Aufsummieren** summiert alle Werte auf.

$\alpha$-**Konfidenzintervall** berechnet ein $\alpha$-Konfidenzintervall für den Mittelwert aus allen Werten. Der Parameter $\alpha$ ist vom Benutzer wählbar.

**Maximum** bestimmt den größten Wert einer Menge von Werten.

**Minimum** bestimmt den kleinsten Wert einer Menge von Werten.

**Median** bestimmt den Median aller Werte.




**Abbildung 11.3.:** Die Statistikeinstellungen der Graphstatistik

**$p$-Quantil** bestimmt das $p$-Quantil aller Werte. Der Parameter $p$ ist vom Benutzer wählbar.

**Standardabweichung** bestimmt die Standardabweichung aller Werte.

**Varianz** bestimmt die Varianz aller Werte.

**Vergleich** stellt alle Werte nebeneinander, zum Beispiel für eine Tabelle.

Der Benutzer kann über den Statistikbereich in der rechten Seite der Graphstatistik (siehe Abbildung 11.3) auswählen, welche der oben aufgelisteten Arten der Zusammenfassung er für Objektmengen und verschiedene Durchläufe möchte. Außerdem kann der das Diagramm wählen, in dem die Statistik dargestellt werden soll (mehr dazu im nächsten Abschnitt).

### 11.2.4. Darstellung von Statistiken

Zur Darstellung der berechneten Werte werden acht Diagrammtypen angeboten, aus denen der Benutzer sich für seine Zwecke geeignete Typen auswählen kann. Zusätzlich können Daten tabellarisch nebeneinander gestellt werden. Die acht unterstützten Diagrammtypen sind:

**Liniendiagramme** stellen einen oder mehrere Datensätze dar. Jeder Datensatz wird durch eine Farbe dargestellt, jeder Eintrag eines Datensatzes wird

zu einem Punkt im Diagramm. Die Punkte der Datensätze werden durch gerade Linien miteinander verbunden.

**Flächendiagramme** entsprechen Liniendiagrammen, allerdings wird bei ihnen die Fläche unterhalb der Linien in der Farbe der Datensätze gezeichnet.

**Kreisdiagramme** stellen einen Satz von Daten dar, in dem die Werte als Anteile an einem Gesamtwert interpretiert werden. Dies könnte z.B. benutzt werden, um die Verteilung von Flusseinheiten auf die Senken darzustellen.

**3D-Kreisdiagramme** verhalten sich genauso wie normale Kreisdiagramme, werden aber im Gegensatz zu diesen dreidimensional als ein flacher Zylinder gezeichnet.

**Ringdiagramme** sind ebenfalls Kreisdiagramme, die als ein Ring gezeichnet werden, an Stelle eines Kreises.

**Säulendiagramme** stellen einen oder mehrere Datensätze dar. Jeder Datensatz wird durch eine Farbe dargestellt und für jeden Eintrag gibt eine Säule in der Farbe des jeweiligen Datensatzes den Wert des Datensatzes an diesem Eintrag an.

**Stufendiagramme** entsprechen in ihren Möglichkeiten Liniendiagrammen, allerdings werden die einzelnen Werte der Datensätze zu Stufen. Es werden also keine Zwischenwerte interpoliert. Dies kann für einige Daten wünschenswert sein, etwa bei Flussraten.

**Stufenflächendiagramme** sind Stufendiagramme, deren Fläche unterhalb der Stufen farbig ausgefüllt ist.

Beispiele für das Aussehen der einzelnen Diagrammtypen sind in den Abbildungen 11.4 und 11.5 zu finden. Der Benutzer kann die Titel und die Achsenbeschriftungen des Diagramms nach seinem Belieben wählen. Dies erfolgt genauso wie das Erstellen, Kopieren, Löschen oder Repositionieren neuer Diagramme über den Diagrammbereich der rechten Seite der Graph-Statistik, wie in Abbildung 11.6 zu sehen.

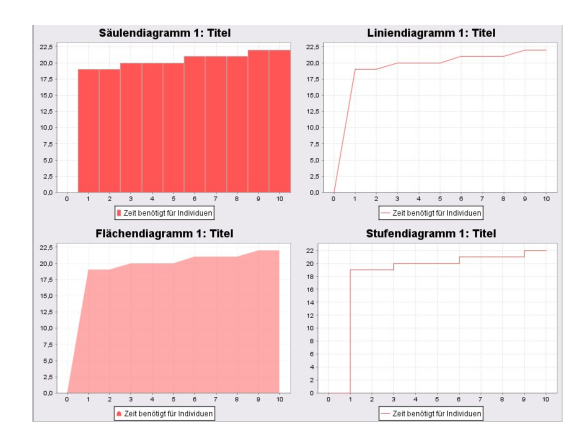



**Abbildung 11.4.:** Säulen-, Linien-, Flächen- und Stufendiagramme

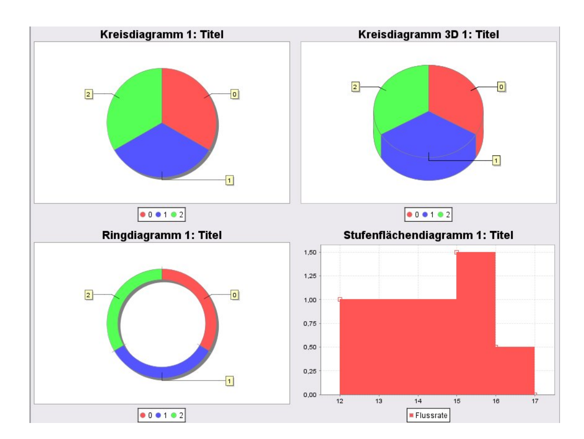



**Abbildung 11.5.:** Kreis-, 3D-Kreis-, Ring- und Stufenflächendiagramme




**Abbildung 11.6.:** Die Diagrammeinstellungen der Graphstatistik

# 12. Visualisierung

## 12.1. Grundlagen

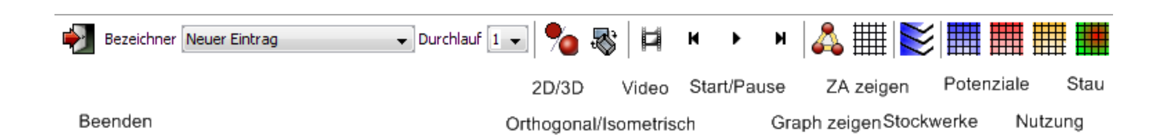



**Abbildung 12.1.:** Symbolleiste der Visualisierung.

Nachdem im Batch eine Simulation bzw. eine Optimierung durchgeführt worden ist, kann das Verhalten der Individuen in der Simulation und auch der berechnete dynamische Fluss betrachtet werden. Der Modus kann über den Reiter VISUALISIERUNG erreicht werden.

Die Visualisierung der Ergebnisse geschieht mittels OpenGL in 3D. Es besteht jedoch auch die Möglichkeit, die Visualisierung in 2D von oben (**Orthogonal**) und in einer Perspektive mit schräger Draufsicht (**Isometrisch**) zu betrachten.

Links in der Symbolleiste befinden sich zwei Auswahlfelder, mit denen sich zum einen die verschiedenen Aufträge aus der Batch-Verarbeitung laden lassen und andererseits innerhalb eines Auftrags die verschiedenen Durchläufe gewechselt werden können. Wenn ein neuer Datensatz geladen wird, wird die zur Visualisierung benötigte Datenstruktur aktualisiert, der Fortschritt wird durch einen Fortschrittsbalken angezeigt.

Nach dem Laden startet die Visualisierung in der Grundstellung, wie in Abbildung 12.2 zu sehen. Falls Graph und Zellulärer Automat erzeugt worden sind, werden beide auch angezeigt. Die Anzeige kann je nach gewählten Darstellungsoptionen abweichen.

In der Symbolleiste befinden sich verschiedene Buttons zum Wechseln der Ansicht, zum Starten und Stoppen der Visualisierung und zum Ein- und Ausblenden verschiedener Objekte. Auf der rechten Seite des Fensters befindet sich eine Eigenschaftenleiste, in der weitere Einstellungen über die sichtbaren Daten getroffen werden können.

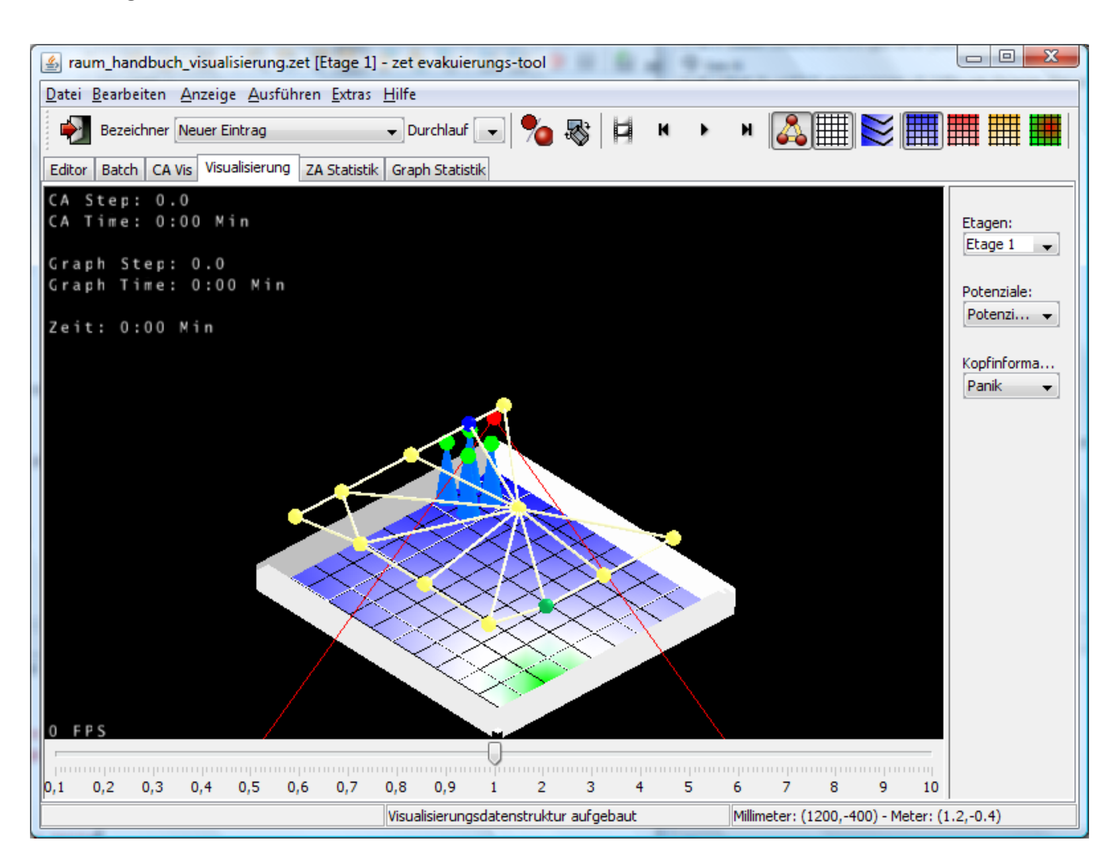



**Abbildung 12.2.:** Grundstellung der Visualisierung.

# 12.2. Eine Visualisierung ansehen

## 12.2.1. Starten und Stoppen

Die Visualisierung kann mit START bzw. PAUSE gestartet und auch pausiert werden. Das Bild auf der Schaltfläche wechselt dabei entsprechend. Mit den danebenliegenden Schalftflächen kann direkt an den Anfang und das Ende gesprungen werden. Die Geschwindigkeit, in der die Simulation abläuft, kann mit dem Schieberegler am unteren Rand geändert werden. Die Bandbreite

reicht dabei von der Verringerung auf ein Zehntel bis zur Beschleunigung auf das Zehnfache.

In der linken oberen Ecke des Visualisierungsbereiches wird die gegenwärtige Zeit sowie der aktuelle Schritt sowohl für den Zellulären Automaten als auch für den Graphen angezeigt. Falls auf einen der beiden Visualisierungsbestandteile verzichtet worden ist, wird auch die Information ausgeblendet.

Mit der Schaltfläche VIDEO wird ein Video der Simulation aufgezeichnet. Mit dem Menüeintrag ANZEIGE || SCREENSHOT oder der Direktzugriffstaste F12 kann ein Screenshot erzeugt werden. Die Screenshots werden im Unterverzeichnis **screenshots** unter dem Dateinamen der Projektdatei und einer anschließenden dreistelligen Nummer im PNG-Format gespeichert.

## 12.2.2. Bedienung der Ansichten

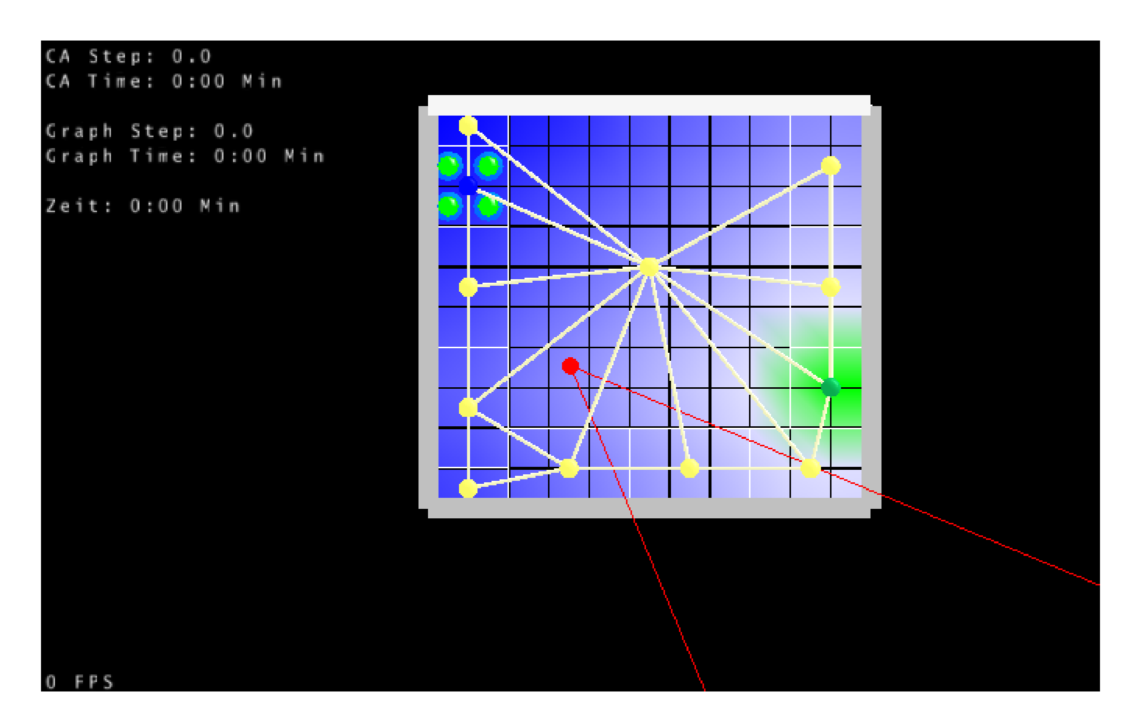



**Abbildung 12.3.:** Orthogonalansicht in der Visualisierung.

Mit dem Button 2D/3D kann generell zwischen der 3-dimensionalen und der 2-dimensionalen Ansicht gewechselt werden. Es ist hierbei zu beachten, dass auch in der 2-dimensionalen Ansicht die Szene 3-dimensional gezeichnet wird, die Projektion jedoch 2-dimensional ist. Die standardmäßig eingestellte Sicht

kann in den Optionseinstellungen zur Visualisierung angepasst werden. Die verschiedenen Sichten laufen synchron, d.h. eine Veränderung der Ansicht in einer Sicht führt auch automatisch zu einer entsprechenden Änderung der Ansicht in den anderen Ansichten (zum Beispiel eine Verschiebung des Standorts). Die Kamera der 3-dimensionalen Sicht ist in den 2-dimensionalen Sichten durch einen roten Punkt angedeutet, ihr Sichtbereich entspricht den zwei roten von diesem Punkt ausgehenden Linien.

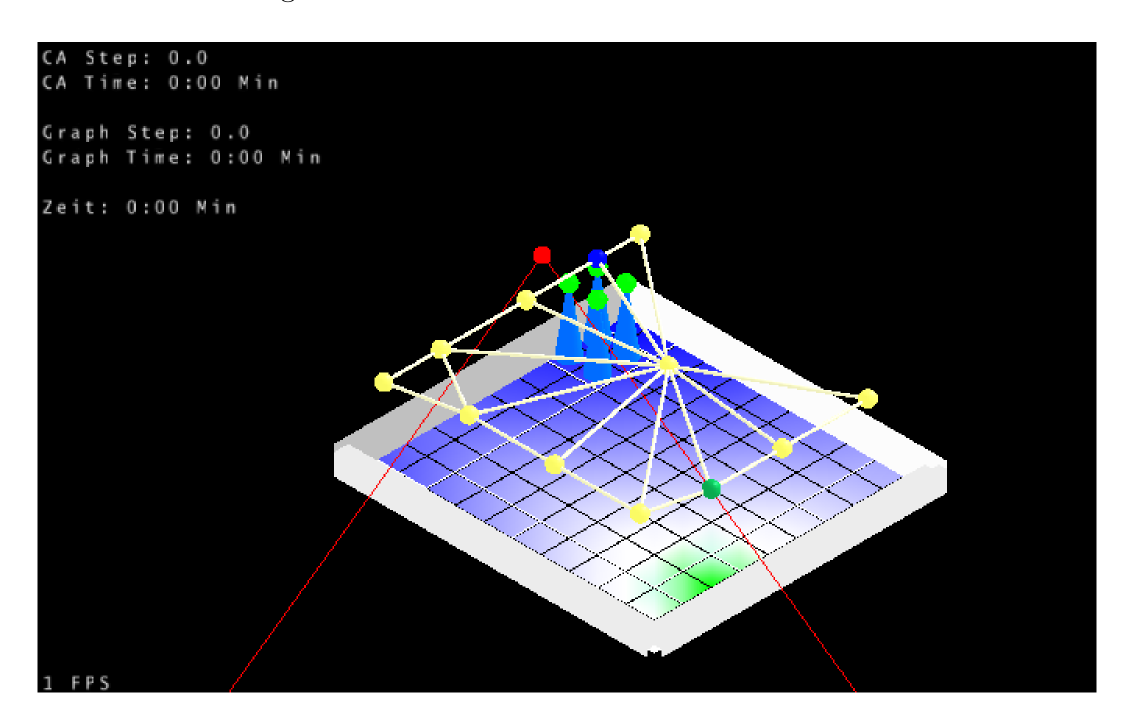



**Abbildung 12.4.:** Isometrische Ansicht in der Visualisierung.

Die 2-dimensionale Sicht lässt sich weiter in zwei verschiedene Projektionen einteilen. In der **orthogonalen** Ansicht wird die Szene direkt von oben, wie auf einem Plan, abgebildet (Abbildung 12.3). Die andere 2-dimensionale Ansicht ist **isometrisch**, d.h. das Gebäude ist schräg von oben zu sehen. In dieser Sicht ist die 3-dimensionale Struktur des Gebäudes sichtbar, parallele Linien bleiben jedoch parallel.

Die Steuerung für diese beiden Sichten ist identisch. Mit den Pfeiltasten kann der sichtbare Bereich des Gebäudes verschoben werden. Mit dem Mausrad oder den Tasten + und - kann näher herangezoomt oder herausgezoomt werden.

Mit einer Einstellung in den Optionen kann zwischen zwei verschiedenen Verhaltensweisen der Maus gewechselt werden. Zunächst einmal kann der Plan

komplett gedreht werden. Im zweiten Modus dreht sich nur die Kamera, der Plan bleibt jedoch immer waagerecht ausgerichtet.

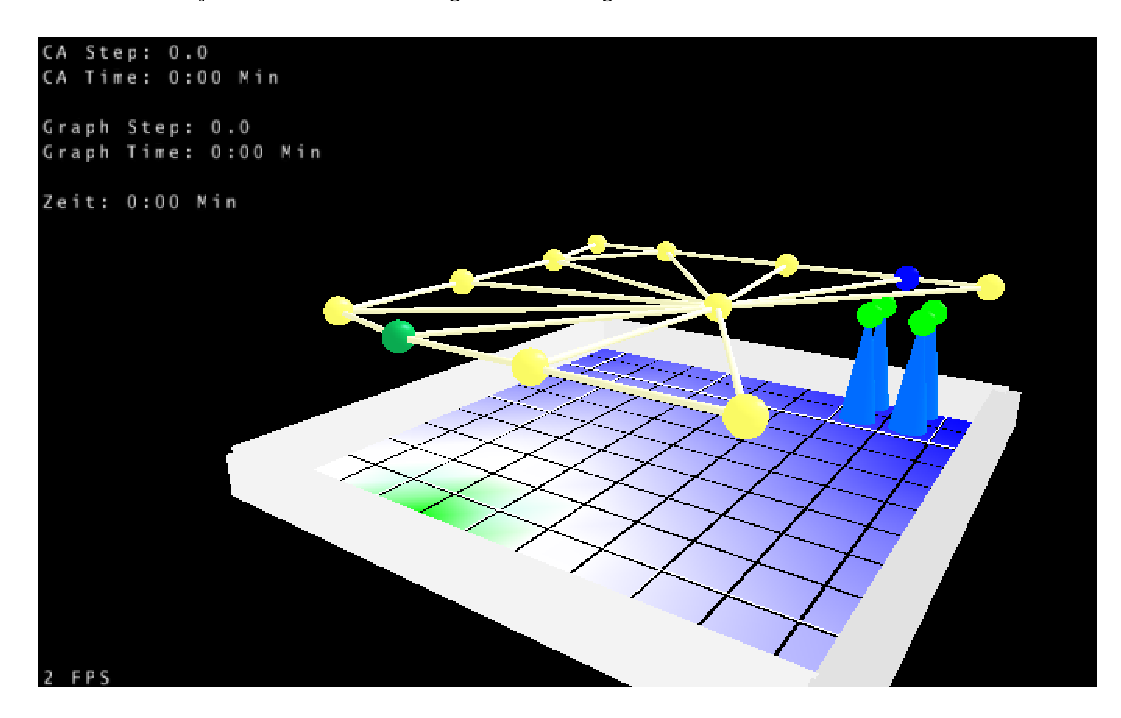



**Abbildung 12.5.:** Die 3-dimensionale Ansicht der Visualisierung.

In der 3-dimensionalen Ansicht erfolgt die Steuerung großteils mit der Maus. Wenn während der Bewegung der Maus die linke Maustaste gedrückt wird, ändert sich auch die Sichtrichtung der Kamera. Drehen des Mausrads bewegt die Kamera nach vorn und hinten. Das gleiche kann mit den Pfeiltasten rauf und runter gemacht werden. Wird die Maus bewegt, während die rechte Maustaste gedrückt gehalten wird, kann die Höhe der Kamera verändert werden.

Mit den Pfeiltasten nach rechts und links wird jedoch nicht die Sichtrichtung geändert, sondern die Kamera macht einen Schritt nach rechts bzw. links. Die Höhe der Kamera kann außerdem noch mit den Tasten + und - eingestellt werden. Die Bewegung nach oben und unten mit der Maus kann in den Optionen invertiert werden.

## 12.3. Die Visualisierung

Mit den Schaltflächen Graph zeigen und ZA zeigen kann sowohl der Graph als auch der zelluläre Automat aus- bzw. eingeblendet werden. Wenn beide

ausgeblendet werden, bleibt nur die Gebäudestruktur in Form von Wänden übrig.

Allgemein können in der Visualisierung einzelne Stockwerke oder das ganze Gebäude dargestellt werden. Mit der Schaltfläche STOCKWERKE in der Symbolleiste kann zwischen der Sicht auf einzelnen Etagen und der Sicht auf das ganze Gebäude umgeschaltet werden. In der Sicht, in der nur eine Etage angezeigt wird, kann mit der ETAGEN-Listenauswahl rechts die angezeigte Etage ausgewählt werden.

### 12.3.1. Die Graphvisualisierung

Zur Graphansicht gehören die Knoten und Kanten des Graphen, die bestimmte Gebiete des Gebäudes repräsentieren. Die Knotenfarbe gibt dabei an, um welche Art Gebiet es sich dabei handelt. Blaue Knoten symbolisieren die Quellen, das heißt die Orte, an denen Personen stehen. Grüne Knoten repräsentieren die Senken, das heißt die Ausgänge. Lilane Knoten sind Knoten, die eigentlich Quellen wären, aber nicht mit der Senke verbunden sind. Alle anderen Knoten sind gelb, ebenso die Kanten.

Die zu den Knoten gehörenden Gebiete werden durch weiße Linien auf dem Boden umrandet, so dass eine Verbindung zwischen der gerasterten Gebäudestruktur und dem Graph hergestellt werden kann. Sobald die Visualisierung gestartet wird, fließt der Fluss in Form von einzelnen, hellblauen Flusseinheiten von den Quellen zu den Senken durch den Graphen.

### 12.3.2. Die Visualisierung des Zellulären Automaten

Zur Ansicht des Zellulären Automaten gehören die Zellen, die Individuen und die Potentiale bzw. verschiedene andere Informationen auf dem Boden der Zellen. Die Farbwerte auf den Zellen können wahlweise in einem weichen Übergang oder scharf abgegrenzt angezeigt werden. Die Ränder der Zellen können ebenfalls durch Linien sichtbar gemacht werden, falls dies gewünscht ist.

Individuen repräsentieren Personen und bestehen aus einem blauen Kegel und einem kugelförmigen Kopf. Der Kopf der Individuen kann auch verschiedene

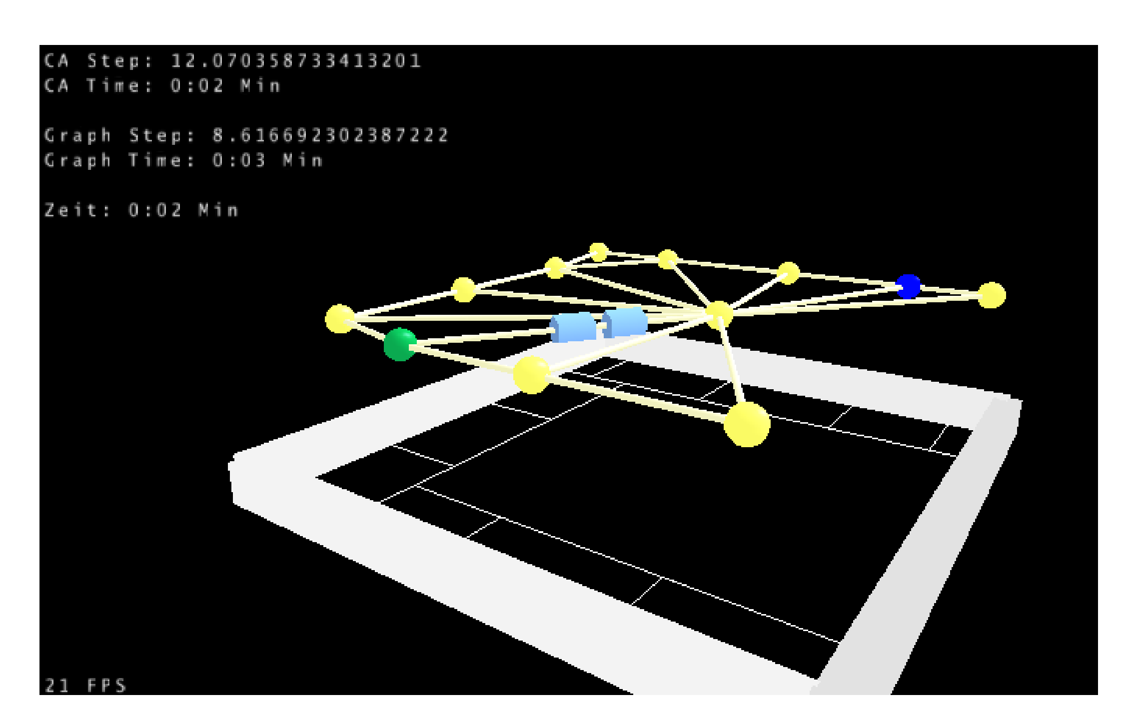



**Abbildung 12.6.:** Visualisierung des Graphen inklusive Fluss.

Statuswerte durch andere Farben darstellen. Die Individuen bewegen sich gemäß der bei der Simulation berechneten Zellübergänge durch das Gebäude.

Die Evakuierungszellen sind grün dargestellt, Treppen werden rosa dargestellt, Sicherheitsbereiche gelb und Verzögerungsbereiche rot. Belegungsgebiete sind nicht weiter gekennzeichnet, ihre Positionen sind durch die Startpositionen der Individuen ersichtlich.

### Informationen auf den Zellen

Falls eine Zelle keinen speziellen Typ hat, können verschiedene Informationen auf den Zellen dargestellt werden. Standardmäßig wird das gemischte statische Potential aller Ausgänge angezeigt. Dabei sind höhere Potentialwerte in dunklerem Blau und niedrigere Potentialwerte in hellerem Blau gekennzeichnet. Die Zellen zeigen also einen Farbverlauf von weiß an den Ausgängen hin zu dunklem Blau an den entferntesten Stellen des Gebäudes. Je dunkler eine Zelle, desto weiter ist sie vom nächstgelegenen Ausgang entfernt.

Die Art der angezeigten Informationen kann durch die vier Schaltflächen Potentiale, Nutzung und Stau ausgewählt werden. Die aktuell ausgewählte Information wird vertieft dargestellt. Wird eine vertiefte Schaltfläche nochmal

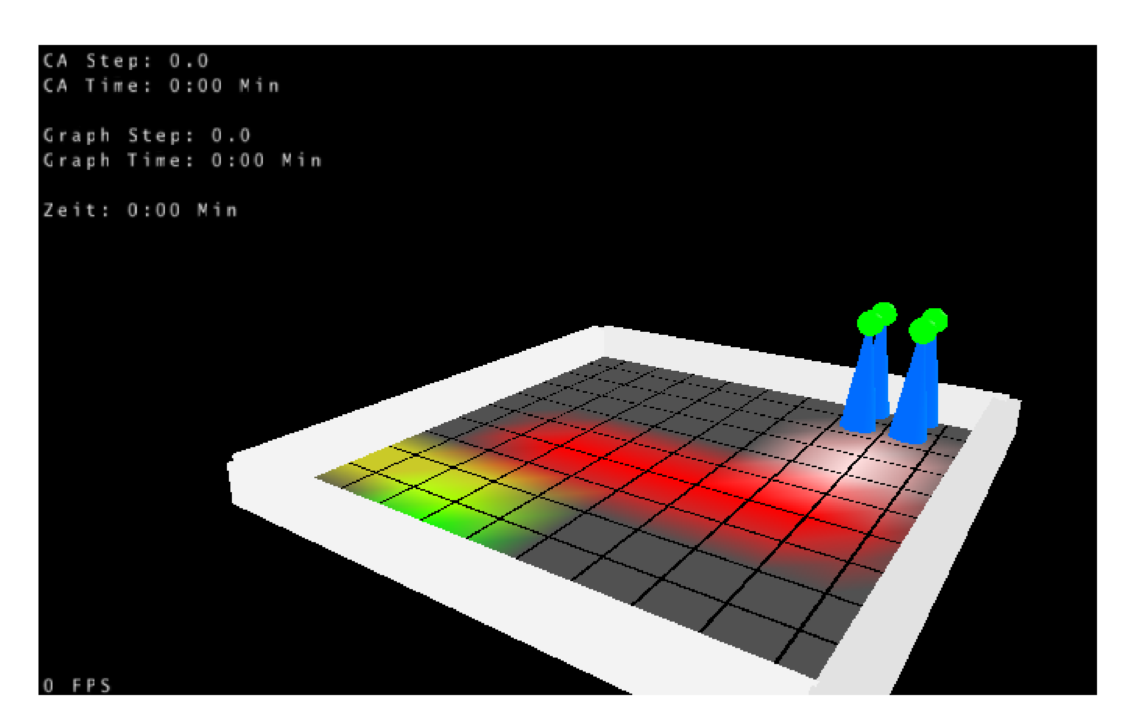



**Abbildung 12.7.:** Visualisierung der Bereiche auf den Bodenzellen.

geklickt, wird keine zusätzliche Information angezeigt und die Zellen erhalten den grauen Standardboden.

Falls mehrere Potentiale vorhanden sind, es also mehrere Ausgänge gibt, können mit der Listenauswahl POTENTIALE im Eigenschaftenbereich rechts einzelne Potentiale angezeigt werden, so dass nicht mehr das minimum über alle Potenziale angezeigt wird.

Der rechte der beiden Schaltflächen für Potentiale ermöglicht das Hinzuschalten des dynamischen Potentials. Das dynamische Potential wird in rot dargestellt und wird dunkler, je stärker es auf einer Zelle ist. Falls es wieder schwächer wird, wird die Zelle schließlich wieder grau. Der Herdentrieb ist also je stärker, desto röter eine Zelle ist.

Die Nutzung der Zellen wird in einem Orangeton angezeigt. Dabei wird eine Zelle stärker eingefärbt, falls sie öfter benutzt worden ist. Damit können häufig verwendete Laufwege der Individuen dargestellt werden.

Der letzte Modus ist der Staumodus. Hier werden die Zellen in einem Farbverlauf von grün nach rot dargestellt, die Farbe hängt von der Anzahl der Individuen ab, die auf einer gegebenen Zelle nicht vorwärtskamen, weil ihre Laufrichtung besetzt war.

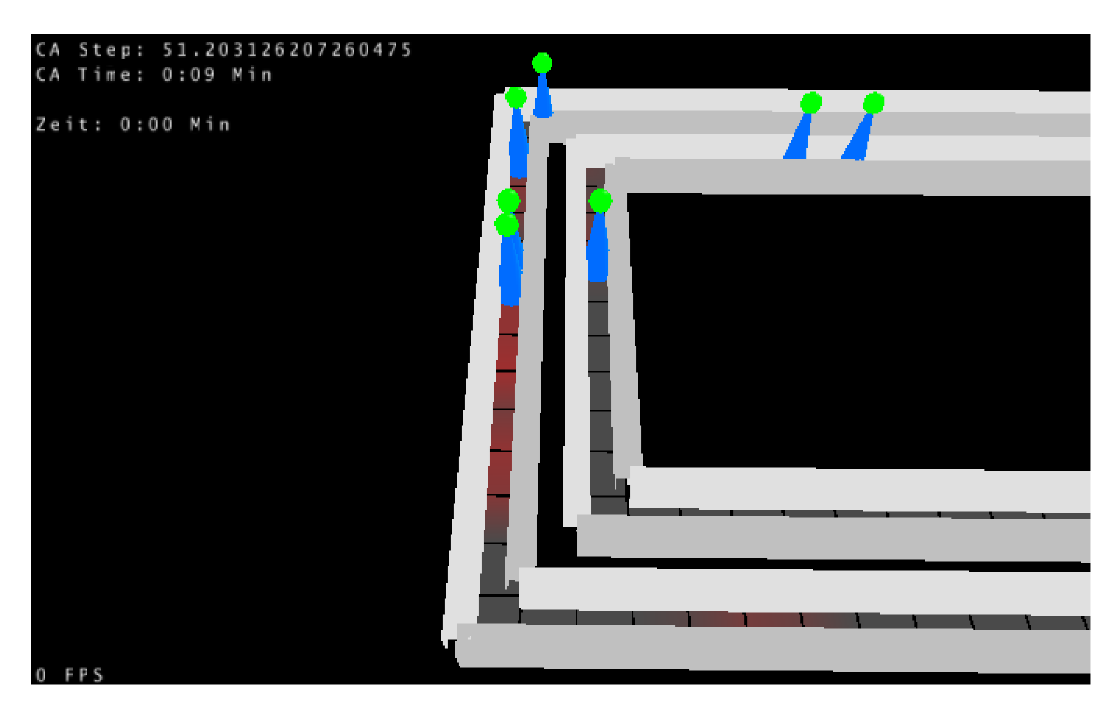



**Abbildung 12.8.:** Individuen in einem Gang hinterlassen ein dynamisches Potenzial.

### Informationen auf den Köpfen

Analog zu den Zellen können auch die Köpfe der Individuen verschiedene Zustände durch einen Farbverlauf symbolisieren. Der gewählte Zustand kann mit dem Listenauswahlfeld KOPFINFORMATION ausgewählt werden. Im Fall **Einfarbig** wird der Kopf in der gleichen Farbe wie der restliche Körper dargestellt.

Falls **Panik** ausgewählt wurde, zeigen die Köpfe mit einem Farbverlauf von grün nach rot die Panik eines Individuums an. Abbildung 12.11 zeigt Individuen an einer Engstelle, die panisch sind. Genauso wird, falls **Erschöpfung** angewählt wird, die Erschöpfung eines Individuums angezeigt. Die Erschöpfung steigt sehr langsam, so dass selten dunkelrote Farbtöne angenommen werden.

Mit **Echte Geschwindigkeit** wird die tatsächlich gelaufene Geschwindigkeit der Individuen ausgewertet. Je langsamer die Individuen vorwärts kommen, desto roter werden sie. Damit können auf den ersten Blick Staustellen erkannt werden.

Zuletzt kann im Modus **Alarmierungszustand** erkannt werden, welche Individuen bereits alarmiert worden sind. Die nicht alarmierten Individuen haben einen roten Kopf, während die bereits alarmierten einen grünen Kopf haben.

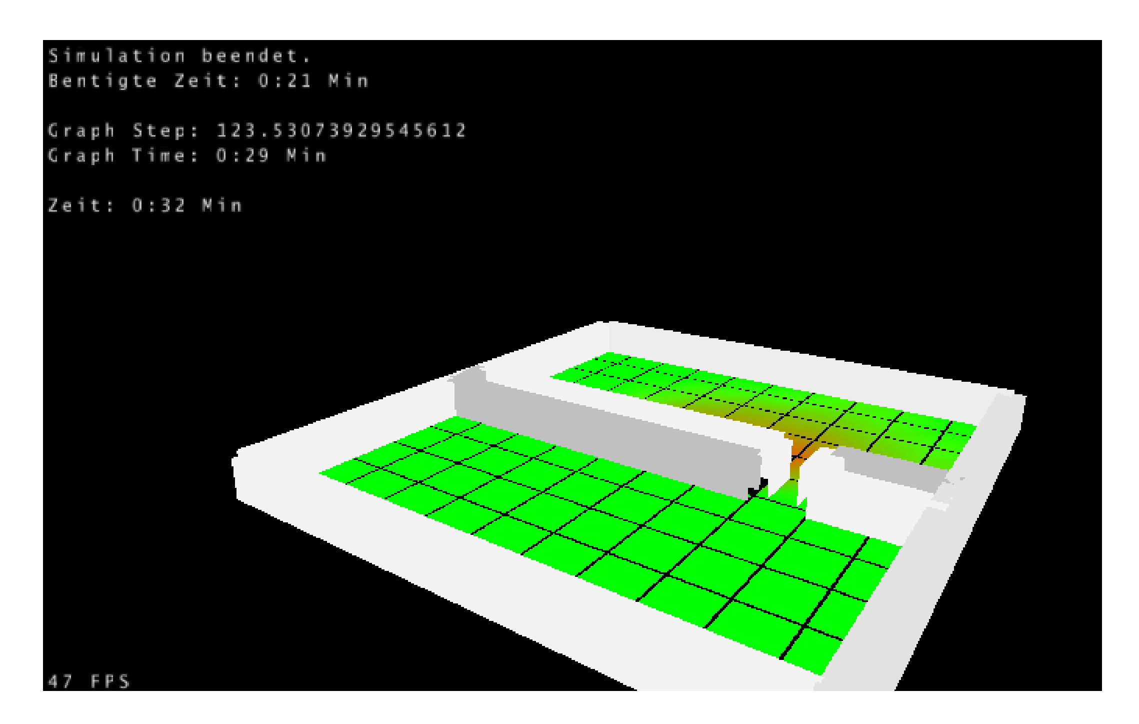



**Abbildung 12.9.:** Eine Engstelle sorgt für leichten Stau und damit rote Zellen.

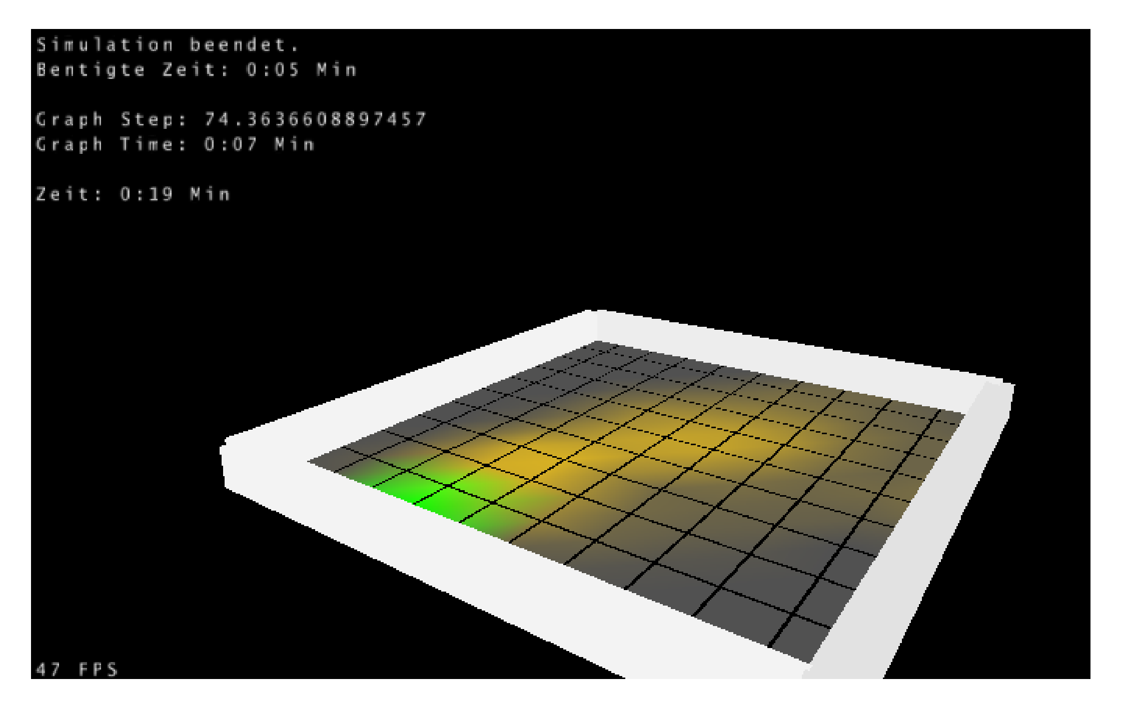



**Abbildung 12.10.:** Visualisierung der Nutzung eines Pfades.

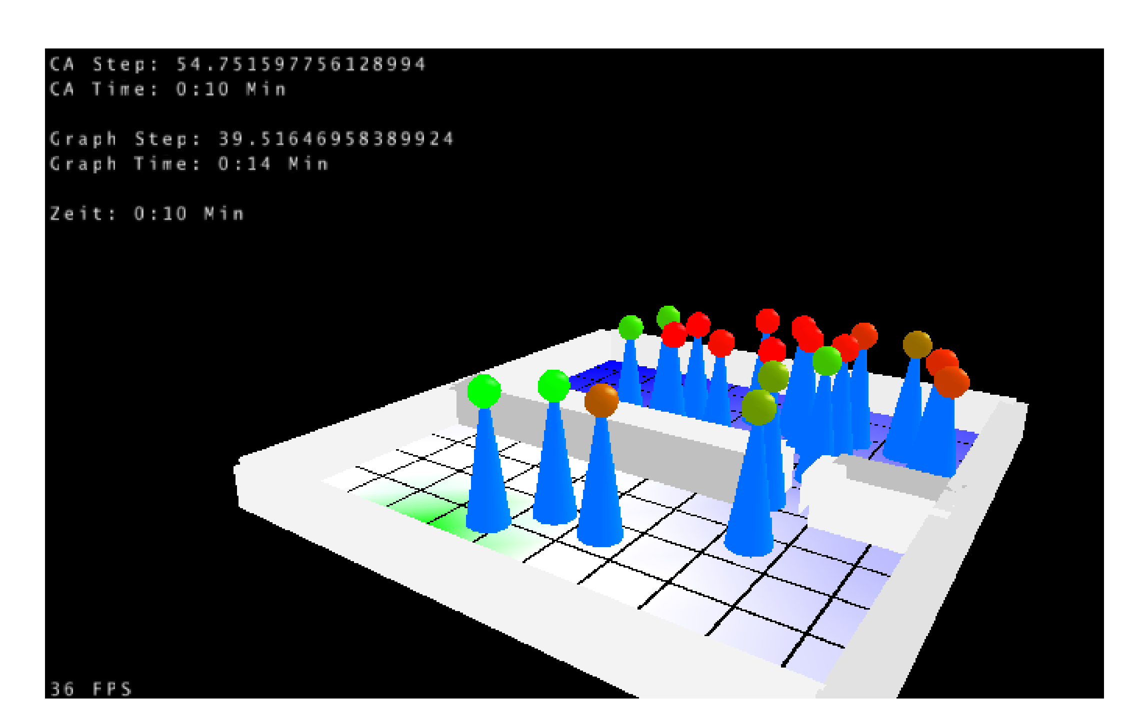



**Abbildung 12.11.:** Panische Individuen an einer Engstelle.

# 13. Weitere Informationen

Das vorliegende Handbuch ist auf dem Stand vom 08.11.2008. Die aktuelle Version des Handbuchs sowie weitere Informationen über **ZET** und die Entstehung der Software finden Sie in Kürze unter der Internetadresse **www.zet-evakuierung.de**.

# Teil V.

# Fazit

## 13.1. Fazit

Während der letzten zwölf Monate haben wir mit **ZET** eine umfangreiche Software zur Untersuchung von Evakuierungen geschrieben. Dabei unterstützen wir einerseits Simulationen mit Hilfe eines Zellulären Automaten und andererseits die Modellierung des zu evakuierenden Gebäudes als Netzwerkflussproblem.

Bei der Umsetzung des Zellulären Automaten haben wir besonders auf Modularität geachtet, so dass das Verhalten der Simulation durch andere Regeln vollständig austauschbar ist. Andererseits haben wir mehrere Regelsätze implementiert und mit Hilfe der Rimea-Richtlinie [1] getestet. Bei der Implementierung der Flussalgorithmen war die Effizienz die größte Herausforderung. Für das für Evakuierungsprobleme besonders interessante Earliest Arrival Transshipment Problem gibt es keine polynomiellen Algorithmen, aber es ist uns gelungen, einen praktisch schnellen Algorithmus zu implementieren [32].

Neben diesen beiden Kernstücken unseres Programms haben wir auch viel Zeit für die Erstellung und Auswertung von Probleminstanzen aufgewendet. Der Editor stellt die notwendigen Funktionalitäten zum Zeichnen von Gebäuden bereit. Eigene Baupläne können dabei transparent eingeblendet werden. Für die automatisierte Durchführung verschiedener Testläufe stellen wir ein Batchfenster bereit. Im Anschluss kann eine statistische Auswertung eingesehen werden. Außerdem stellt **ZET** umfangreiche Visualisierungsmöglichkeiten der Ergebnisse bereit. Dabei können sowohl Simulationen als auch berechnete Flüsse angezeigt werden, wobei die Darstellung mit Hilfe einer Vielzahl von Einstellungsmöglichkeiten angepasst werden kann. Die Visualisierung kann auch als Video exportiert werden.

**ZET** ist also eine Rundum-Software: Man kann damit Gebäude zeichnen, Evakuierungen simulieren, optimieren, auswerten und visualisieren.

Natürlich gibt es trotz der vielen positiven Punkte auch Ziele, die wir noch gern erreichen würden. Darunter fallen zum Beispiel Ansätze zur automatischen Generierung guter Fluchtpläne, ein sehr interessanter und komplizierter Themenbereich. Für weitere Entwicklungen und Erweiterungen unserer Software möchten wir das Projekt gerne als Open-Source-Projekt veröffentlichen. Informationen darüber, wie es mit **ZET** weitergeht, gibt es in Kürze unter **www.zet-evakuierung.de**.

# Abbildungsverzeichnis

# Literaturverzeichnis

[1] *Richtlinie für Mikroskopische Entfluchtungsanalysen*, Nummer 2.1.0, 2008. www.rimea.de.

[2] Ahuja, Ravindra K., Thomas L. Magnanti und James B. Orlin: *Network Flows. Theory, Algorithms, and Applications.* Prentice Hall, 1993.

[3] Algorithmic Solutions, Software GmbH: *LEDA.* http://www.algorithmic-solutions.com/.

[4] Baumann, Nadine und Ekkehard Köhler: *Approximating earliest arrival flows with flow-dependent transit times.* Discrete Applied Mathematics, 155(2):161–171, 2007.

[5] Baumann, Nadine und Martin Skutella: *Solving evacuation problems efficiently: Earliest arrival flows with multiple sources.* Seiten 399–408, 2006.

[6] Busacker, Robert G. und Paul J. Gowen: *A procedure for determining minimal-cost network flow patterns.* Technischer Bericht 15, Operational Research Office, Johns Hopkins University, 1961.

[7] Carey, Malachy und Eswaran Subrahmanian: *An approach to modeling time-varying flows on congested networks.* Transportation Research Part B, 34:157–183, 2000.

[8] Cherkassky, Boris V. und Andrew V. Goldberg: *On implementing push-relabel method for the maximum flow problem.* Algorithmica, 19:390–410, 1997.

[9] Choi, W, R. L. Francis, Horst W. Hamacher und Suleyman Tufekci: *Network models of building evacuation problems with flow dependent exit capacities.* Seiten 1047–1059, 1984.

[10] Choi, W., Horst W. Hamacher und Suleyman Tufekci: *Modeling of building evacuation problems by network flows with side constraints.* European Journal of Operational Research, 35:98–110, 1988.

[11] Edmonds, Jack und Richard M. Karp: *Theoretical improvements in algorithm efficiency for network flow problems.* Journal of the ACM, 19:248–264, 1972.

[12] Ford, Lester R. und Delbert R. Fulkerson: *Flows in Networks.* Princeton University Press, 1962.

[13] Goldberg, Andrew V. und Robert E. Tarjan: *Finding minimum-cost circulations by cancelling negative cycles.* Journal of the ACM, 36:873–886, 1989.

[14] Gwynne, Steve, Ed R. Galea, Matt Owen, Peter J. Lawrence und Lazaros Filippidis: *A review of the methodologies used in computer simulation of evacuation from the built environment.* Building and Environment, 34:741–749, 1999.

[15] Hall, Alex, Katharina Langkau und Martin Skutella: *An FPTAS for Quickest Multicommodity Flows with Inflow-Dependent Transit Times.* Algorithmica, 47(3):299–321, 2007.

[16] Hamacher, Horst W. und Stevanus A. Tjandra: *Mathematical Modelling of Evacuation Problems: A State of the Art.* Pedestrian and Evacuation Dynamics, Seiten 227–266, 2002.

[17] Helbing, Dirk, Illés Farkas und Tamás Vicsek: *Simulating dynamical features of escape panic.* Nature, (407):487 – 490, 2000.

[18] Hoppe, Bruce E.: *Efficient Dynamic Network Flow Algorithms.* Doktorarbeit, 1995.

[19] Köhler, Ekkehard, Katharina Langkau und Martin Skutella: *Time-expanded graphs for flow-dependent transit times.* Band 2461, Seiten 599–611. LNCS, 2002.

[20] Köhler, Ekkehard und Martin Skutella: *Flows over Time with Load-Dependent Transit Times.* SIAM Journal on Optimization, 15(4):1185–1202, 2005.

[21] Klüpfel, Hubert: *A Cellular Automaton Model for Crowd Movement and Egress Simulation.* 2003.

[22] Korte, Bernhard und Jens Vygen: *Combinatorial Optimization.* Springer, 1. Auflage, 2001.

[23] Korte, Bernhard und Jens Vygen: *Combinatorial Optimization*, Seiten 195–198. Springer, 3. Auflage, 2006.

[24] Korte, Bernhard und Jens Vygen: *Combinatorial Optimization*, Seiten 199–202. Springer, 3. Auflage, 2006.

[25] Langkau, Katharina: *Flows over Time with flow dependent transit times.* Technische Universität Berlin, Fachbereich Mathematik, 2003.

[26] Minieka, Edward: *Maximal, lexicographic, and dynamic network flows.* Operations Research, 21:517–527, 1973.

[27] Raney, Bryan K.: *Learning Framework for Large-Scale Multi-Agent Simulations.* Doktorarbeit, 2005. http://matsim.org.

[28] Rauf, Imran: *Earliest Arrival Flows with Multiple Sources.* Diplomarbeit, 2005.

[29] Schreckenberg, Michael und Som D. Sharma: *Pedestrian and Evacuation Dynamics.* 2002.

[30] Seyfried, Armin, Bernhard Steffen und Thomas Lippert: *Basics of modelling the pedestrian flow.* Physica A, 368:232–238, 2006.

[31] Tardos, Éva: *A strongly polynomial minimum cost circulation algorithm.* Combinatorica, 5:247–255, 1985.

[32] Tjandra, Stevanus A.: *Dynamic Network Optimization with Application to the Evacuation Problem.* Shaker, 1. Auflage, 2003.